By Dirk Krafzig, Karl Banke, Dirk Slama

Enterprise SOA: Service-Oriented Architecture Best Practices

#### Team LiB



Table of Contents
 Index

• Index

Publisher: Prentice Hall PTR Pub Date: November 09, 2004 ISBN: 0-13-146575-9 Pages: 408

Enterprise SOA presents a complete roadmap for leveraging the principles of S Architectures to reduce cost and risk, improve efficiency and agility, and liberat the vagaries of changing technology.

NEXT 🕨

NEXT 🕨

- Benefit from the lessons of four enterprise-level SOA case studies from of Scotland, and other world-class enterprises
- Make your business technology independent and manage infrastructure on architecture, not specific implementation techniques
- Recognize the technical and nontechnical success factors for SOA in the
- Define and communicate the economic value proposition of an SOA
- Apply pragmatic design principles to solve the problems of data and pro environment

Whether you're a manager, architect, analyst, or developer, if you must drive g services, Enterprise SOA will show you howfrom start to finish.

#### Team LiB



#### Team LiB

## ENTERPRISE SOA Dan Rasta Kata Barne

Index

Table of Contents

Enterprise SOA: Service-Oriented Architecture Best Practices

By Dirk Krafzig, Karl Banke, Dirk Slama



Copyright

Publisher: Prentice Hall PTR Pub Date: November 09, 2004 ISBN: 0-13-146575-9 Pages: 408

◀ PREVIOUS NEXT ▶

#### Praise for Enterprise SOA The Coad Series Acknowledgments About the Authors Dirk Krafzig Karl Banke Dirk Slama Foreword Reader's Guide Who Should Read This Book A Roadmap for This Book Chapter 1. An Enterprise IT Renovation Roadmap Section 1.1. Agony Versus Agility Section 1.2. Enterprise Software Is a Different Animal Section 1.3. The Importance of Enterprise Software Architectures Section 1.4. The Requirements for an Enterprise Software Architecture Section 1.5. The Relation of Enterprise Architecture and Enterprise Standards Section 1.6. Organizational Aspects Section 1.7. Lifelong Learning Section 1.8. The Enterprise IT Renovation Roadmap Chapter 2. Evolution of the Service Concept Section 2.1. Milestones of Enterprise Computing Section 2.2. Programming Paradigms Section 2.3. Distributed Computing Section 2.4. Business Computing Section 2.5. Conclusion Chapter 3. Inventory of Distributed Computing Concepts Section 3.1. Heterogeneity of Communication Mechanisms Section 3.2. Communication Middleware Section 3.3. Synchrony Section 3.4. Interface Versus Payload Semantics Section 3.5. Tight Versus Loose Coupling Section 3.6. Conclusion Part I. Architectural Roadmap Chapter 4. Service-Oriented Architectures Section 4.1. What Is a Software Architecture? Section 4.2. What Is a Service-Oriented Architecture? Section 4.3. Elements of a Service-Oriented Architecture Section 4.4. Conclusion Chapter 5. Services as Building Blocks Section 5.1. Service Types Section 5.2. Layers on the Enterprise Level Section 5.3. Conclusion Chapter 6. The Architectural Roadmap Section 6.1. The Architectural Roadmap Section 6.2. Fundamental SOA Section 6.3. Networked SOA Section 6.4. Process-Enabled SOA Section 6.5. Conclusion Chapter 7. SOA and Business Process Management Section 7.1. Introduction to BPM Section 7.2. BPM and the Process-Enabled SOA Section 7.3. Conclusion Chapter 8. Managing Process Integrity Section 8.1. Data Versus Process Integrity Section 8.2. Technical Concepts and Solutions

Section 8.3. Recommendations for SOA Architects Section 8.4. Conclusion Chapter 9. Infrastructure of the Service Bus Section 9.1. Software Buses and the Service Bus Section 9.2. Logging and Auditing Section 9.3. Availability and Scalability Section 9.4. Securing SOAs Section 9.5. Conclusion Chapter 10. SOA in Action Section 10.1. Building Web Applications Section 10.2. Enterprise Application Integration Section 10.3. Business-to-Business Section 10.4. Fat Clients Section 10.5. Designing for Small Devices Section 10.6. Multi-Channel Applications Section 10.7. Conclusion Part II. Organizational Roadmap Chapter 11. Motivation and Benefits Section 11.1. The Enterprise Perspective Section 11.2. The Personal Perspective Section 11.3. Conclusion Chapter 12. The Organizational SOA Roadmap Section 12.1. Stakeholders and Potential Conflicts of Interest Section 12.2. The Organizational SOA Roadmap Section 12.3. Four Pillars for Success Section 12.4. An Ideal World Section 12.5. The Real WorldOrganization-Wide Standards Section 12.6. Recommendations for the SOA Protagonist Section 12.7. Conclusion Chapter 13. SOA-Driven Project Management Section 13.1. Established Project Management Methodologies Section 13.2. SOA-Driven Project Management Section 13.3. Configuration Management Section 13.4. Testing Section 13.5. Conclusion Part III. Real-World Experience Chapter 14. Deutsche Post AG Case Study Section 14.1. Project Scope Section 14.2. Implementation Section 14.3. Technology Section 14.4. Lessons Learned, Benefits, and Perspectives Chapter 15. Winterthur Case Study Section 15.1. Project Scope Section 15.2. Implementation Section 15.3. Technology Section 15.4. Lessons Learned, Benefits, and Perspectives Chapter 16. Credit Suisse Case Study Section 16.1. Project Scope Section 16.2. Implementation Section 16.3. Technology Section 16.4. Lessons Learned, Benefits, and Perspectives Chapter 17. Halifax Bank Of Scotland: IF.com Section 17.1. Project Scope Section 17.2. Implementation Section 17.3. Technology Section 17.4. Lessons Learned, Benefits, and Perspectives

```
Index
```

Team LiB



Team LiB

## Copyright

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Publisher: John Wait

Editor in Chief: Don O'Hagan

Executive Editor: Paul Petralia

Editorial Assistant: Michelle Vincenti

Marketing Manager: Chris Guzikowski

Cover Designer: Jerry Votta

Managing Editor: Gina Kanouse

Project Editor: Christy Hackerd

Copy Editor: Benjamin Lawson

Indexer: Lisa Stumpf

Compositor: Mary Sudul

Manufacturing Buyer: Dan Uhrig

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U. S. Corporate and Government Sales (800) 382-3419 corpsales@pearsontechgroup.com

For sales outside the U. S., please contact:

International Sales international@pearsoned.com

Visit us on the Web: www.phptr.com

Library of Congress Cataloging-in-Publication Data:

### 2004110321

Copyright © 2005 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc. Rights and Contracts Department One Lake Street Upper Saddle River, NJ 07458

Text printed in the United States on recycled paper at *Phoenix BookTech in Hagerstown, Maryland* 

First printing, November, 2004

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

♦ PREVIOUS NEXT ►

## Praise for Enterprise SOA

"As we started with the development of the e-Platform at the Winterthur, we knew that there were still many questions to be answered. Today, we can look back at a process, which has created the corresponding architectural guidelines, processes, and infrastructure components. In the meantime, we are reaping the benefits of our strategy and are transferring, step by step, our traditional application landscape into a loosely coupled SOA. This forms, as well, the basis for our next step in the direction of Business Process Management. This book clearly describes the many concepts that we painstakingly developed at that time and answers the most important questions that are encountered on the way to an adaptable application landscape for large-scale enterprises. From my point of view, this is a book that should be read by all those who are considering remodeling their application landscape."

#### Daniele Lisetto, Head Technical and Application Platforms, Winterthur Group

"Enterprise SOA provides strategies that help large enterprises to increase the agility of their IT systemsone of the most pressing issues of contemporary IT. Covering both a business and architectural view, these strategies aim to promote the implementation of an IT infrastructure that can serve as a base for the development of truly flexible business processes. This book covers its subject with great profoundness based on real-world evidence. It is in the interest of everybody involved with software architectureparticularly for anybody who intends to establish a Service-Oriented Architectureto read this book."

#### Dr. Helge Heß, Director Business Process Management, IDS Scheer AG

"The SOA principles described in this book are the foundation on which enterprises can build an IT architecture that will satisfy today's most important IT requirements agility and flexibility affordable costs."

### Martin Frick, Head of IT, Winterthur Group

"By delivering SAP's next-generation applications based on a Service-Oriented Architecture, SAP is at the forefront of making Web services work for the enterprise. The Enterprise Services Architecture enables unprecedented flexibility in business process deployment, allowing companies to execute and innovate end-to-end processes across departments and companies, with minimum disruption to other systems and existing IT investments. This strategy comes to life with SAP NetWeaver, which is the technological foundation of the Enterprise Services Architecture. It provides easy integration of people, information, and systems in heterogeneous IT environments and provides a future proof application platform. *Enterprise SOA* provides readers with the architectural blueprints and SOA-driven project management strategies that are required to successfully adopt SOA on an enterprise level."

#### Dr. Peter Graf, SVP Product Marketing, SAP

The SOA principles outlined in this book enable enterprises to leverage robust and proven middleware platforms, including CORBA, to build flexible and business-oriented service architectures. The authors also clearly describe the right strategies for using Model Driven Architecture (MDA) to manage SOA Service Repositories in a platform-independent way, enabling enterprises to better address the problem of heterogeneity at many levels. The Object Management Group was created just to address this central problem of integration in the face of constantly changing heterogeneity and platform churn, so I strongly recommend this book for the bookshelf of every enterprise architect and developer.

Richard Mark Soley, Ph.D. Chairman and Chief Executive Officer, Object Management Group, Inc.

Team LiĐ



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

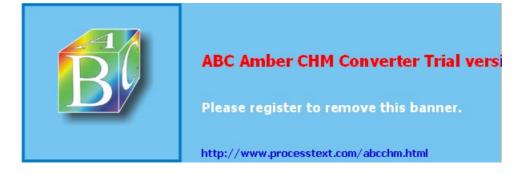
◀ PREVIOUS NEXT ►

## **The Coad Series**

Peter Coad, Series Editor

- David J. Anderson Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results
- David Astels Test Driven Development: A Practical Guide
- David Astels, Granville Miller, Miroslav Novak A Practical Guide to eXtreme Programming
- Andy Carmichael, Dan Haywood Better Software Faster
- Donald Kranz, Ronald J. Norman A Practical Guide to Agile Unified Process
- James McGovern, Scott W. Ambler, Michael E. Stevens, James Linn, Vikas Sharan, Elias Jo *A Practical Guide to Enterprise Architecture*
- Jill Nicola, Mark Mayfield, Michael Abney Streamlined Object Modeling: Patterns, Rules, and Implementation
- Stephen R. Palmer, John M. Felsing A Practical Guide to Feature-Driven Development

#### Team LiB



## **Acknowledgments**

Many people have contributed to this book in many ways, both directly and indirectly over a period of two years. It is therefore impossible to list everybody who influenced, supported, or aided us on our journey from the initial idea to the final manuscript. Nevertheless, we owe our gratitude to at least the following people:

We would like to thank our publisher, Pearson Education, for the support that led to this book being completed. In particular, we would like to thank Paul Petralia, our editor, and Christy Hackerd.

The case studies in this book would not have been possible without extensive cooperation with SOA protagonists from different organizations. We would like to thank Uwe Bath (Deutsche Post), Fiorenzo Maletta (Winterthur), Claus Hagen (Credit Suisse), and Willie Nisbet and Allan Kelly (Halifax Bank of Scotland).

This book would also not have been possible without countless discussions and critical feedback of colleagues, friends, and reviewers. We extend our gratitude to Beat Aeschlimann, Arne Allee, Michael Aschenbrenner, Arnaud Blandin, Ndome Cacutalua, Frank Eversz, Dietmar Grubert, Stefan Havenstein, Georgia and Paul Hickey, Stefan Krahm, James McGovern, Dirk Marwinski, Manfred Mayer, Steve Morris, Ingrid Müller, Joanis Papanagnu, Arnold Pott, Kai Rannenberg, Kirsten Scharf, Uli Steck, Paul Stemmet, Michael Stevens, Harald Störrle, Peter Stracke, Josef Wagenhuber, and Frank Werres.

Sebastian-Svante Wellershoff contributed to the design of our artwork. Birgit Anders supported the author-team by proofreading, formatting, drawing figures, and many other activities that were necessary to produce a manuscript for this book. We all appreciated the accurate and swift manner in which she worked. Mark Roantree was our proofreader for linguistic and some of the technical aspects of this book.

Roland Tritsch and Joachim Quantz have both closely worked with the author team and developed many of the ideas that were presented in this book. They were like a virtual extension of the author team.

Last but not least, we owe our special gratitude to our partners, families, and friends for their patience, time, support, and encouragement. We know that we cannot make it up to you.

Team LiB



Team LiB

▲ PREVIOUS NEXT ►

## **About the Authors**

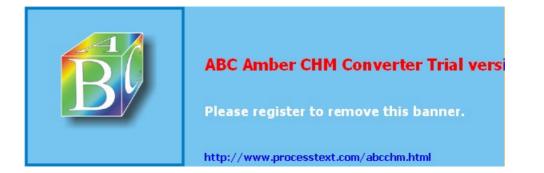
Dirk Krafzig

Karl Banke

Dirk Slama

Team LiB

◀ PREVIOUS NEXT ▶



## **Dirk Krafzig**

Dirk has been dealing with the challenges of enterprise IT and distributed software architectures throughout his entire working life. He devoted himself to SOA in 2001 when he joined Shinka Technologies, a start-up company and platform vendor in the early days of XML-based Web services. Since then, Dirk has acquired a rich set of real world experience with this upcoming new paradigm both from the view point of a platform vendor and from the perspective of software projects in different industry verticals.

Writing this book was an issue of personal concern to him as it provided the opportunity to share his experiences and many insights into the nature of enterprise IT with his readers.

Today, Dirk is designing enterprise applications and managing projects, applying the guiding principles outlined in this book. Dirk has a Ph.D. in Natural Science and an MSc in Computer Science. He lives in Düsseldorf, Germany, and is 39 years old, married, and the father of two children.

#### Team LiB



## Karl Banke

Software architecture has been with Karl since he programmed his first TRON-like game on the then state-of-the art ZX81 in the early 1980s. After graduating as a Master of Physics, he gained his commercial experience in various consulting assignments, mostly in the financial and telecommunications sector.

He moved through stages of consultant, technical lead, software architect, and project manager using a variety of object-oriented technologies, programming languages, and distributed computing environments. Soon realizing that he was too constrained as an employee in doing what he thought necessary in software development, he co-founded the company iternum in 2000, where he currently acts as a principal consultant and general manager.

Karl permanently lives in Mainz, Germany when not temporarily relocated by a current project.

#### Team LiB



## **Dirk Slama**

Having spent the last ten years at the forefront of distributed computing technology, Dirk has developed an in-depth understanding of enterprise software architectures and their application in a variety of industry verticals. Dirk was a senior consultant with IONA Technologies, working with Fortune 500 customers in Europe, America, and Asia on large-scale software integration projects. After this, Dirk set up his own company, Shinka Technologies, which successfully developed one of the first XML-based Web services middleware products, starting as early as 1999.

Dirk holds an MSc in computer sciences from TU-Berlin and an MBA from IMD in Lausanne. He is a co-author of *Enterprise CORBA* (Prentice Hall, 1999), the leading book on CORBA-based system architectures. Dirk is currently working as a solution architect for Computer Sciences Corporation in Zurich, Switzerland.

## Contact: authors@enterprise-soa.com

### Team LiĐ



Team LiB

## Foreword

At the turn of the nineteenth century, a wave of new technologies such as the steam engine, electricity, the loom, the railway, and the telephone emerged. Urbanization and the mass production of goods in large factories fundamentally changed how mankind lived and worked together.

One hundred years later, the industrial revolution had not slowed down: At the turn of the twentieth century, automation, specialization, and a never-ending spiral of efficiency improvement have resulted in modern economies with unheard-of industrial productivity.

After a phase of consolidation during the transition from the twentieth to the twenty-first century, globalization and virtualization have now become the key drivers of our economic lives. Without a doubt, they will yet again change how we live and work together.

If we take a closer look at the past 20 years, we can observe that established business rules have been constantly redefined. New business models emerged; small companies quickly grew into billion-dollar multinationals, aggressively attacking other established companies. A wave of mergers, acquisitions, and buyouts changed the overall industrial landscape.

IT has played a major role in all of this, be it through controlling production processes and supply chains or by creating real-time links between financial markets, thus virtually eliminating arbitrage opportunities by closing the time gaps of trading around the globe. The Internet boom and the "virtual enterprise" are cornerstones of this ongoing development. Entirely new products and services have been created, which would have been unthinkable without the support of modern IT.

Without a doubt, today's modern enterprises are completely dependent on their IT. Consequently, today's IT is driven by the same dynamics as the enterprise itself. Today, we expect an extremely high level of flexibility and agility from our enterprise IT. During the post Internet-boom years, cost efficiency quickly became another key requirement, if not the most important one.

Enterprise IT has changed as a result of the constantly increasing pressure. In the early days of enterprise computing, IT was merely responsible for providing storage and processing capacity, with more and more business logic being added throughout the decades. During the different boom phases in the 1980s and 1990s, a plethora of new applications emerged, often side by side with the information silos that had been developed in the previous 20 years.

Today, the increasing cost pressure is forcing us to efficiently reuse existing systems while also developing new functionality and constantly adapting to changing business requirements. The term "legacy system" is now often replaced with "heritage system" in order to emphasize the value that lies in the existing systems.

The increases in reuse and harmonization requirements have been fueled by the urgency of integrating the historically grown IT landscapes in order to improve IT efficiency and agility. As a result, we could observe at a technical level the emergence of middleware tools and Enterprise Application Integration (EAI) platforms in what can be seen as a post-RDBMS phase.

While a lot of trial-and-error projects were executed in the 1990s, with more or less high levels of success, the development of EAI and middleware concepts has now been culminated in the principles of Service-Oriented Architecture (SOA), which can be seen as an important evolutionary point in the development of integration technologies.

What is important about SOA is that it has taken away the focus from fine-grained, technology-oriented entities such as database rows or Java objects, focusing instead on business-centric services with business-level transaction granularity. Furthermore, SOA is not an enterprise technology standard, meaning it is not dependent on a single technical protocol such as IIOP or SOAP. Instead, it represents an architectural blueprint, which can incorporate many different technologies and does not require specific protocols or bridging technologies. The focus is on defining cleanly cut service contracts with a clear business orientation.

. . . . . . . . . . . . . .

### Team LiB



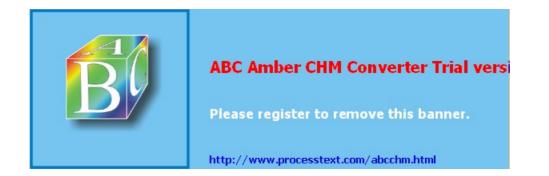
## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

## **Reader's Guide**

The reader's guide provides an indication as to who should read this book and the benefits to be gained. A summary of each chapter provides an overview of the step-by-step approach required for the successful introduction of Service-Oriented Architectures (SOA).

#### Team LiB



Team LiB

## Who Should Read This Book

This book is aimed at the various stakeholders of enterprise software architectures, including software architects and evangelists, designers, analysts, developers, members of IT strategy departments, project managers, representatives of product vendors, and those interested in software architecture and its relation to structures and processes within large-scale organizations. Furthermore, this book is an excellent introduction to the real world of commercial computing for students in a variety of disciplines.

If you are a **software architect**, this book provides you with hands-on guidelines for the design of SOAs. You will find the definition of an SOA together with its key terms as we distinguish the SOA from approaches such as component architectures and software buses. Furthermore, this book provides concrete guidance for the most important design decisions one will encounter in practice. These guidelines comprise identifying services, assigning the appropriate service type and allocating the ownership of data to services. You will also discover how to utilize expansion stages in order to enable stepwise SOA introduction. This book also provides valuable advice on the design of a functional infrastructure for business processes and on how to achieve process integrity, approach heterogeneity, and initiate the technical infrastructure. We discuss these guidelines with respect to different application types, including Web applications, fat clients, mobile applications, EAI, and multi-channel applications. For the purpose of software architects, <u>Chapters 4</u> to <u>10</u> are most valuable. In addition, <u>Chapter 13</u>, which covers SOA project management, will be helpful in ensuring an efficient collaboration within an SOA project. Finally, the case studies in <u>Part III</u> give you practical examples of how architects in other organizations introduced an SOA.

Do you see yourself in the role of an SOA **evangelist**? If you intend to implement an SOA within your own organization, you must successfully promote your ideas. Most importantly, you must be able to communicate the benefits of the SOA to all stakeholders of the application landscape within your organization. <u>Chapter 11</u> will be of special interest to you because it presents the key benefits of SOA for the organization and each individual stakeholder. In addition, <u>Chapter 12</u> provides an in-depth description of the steps required to set up an SOA, with considerable practice-oriented advice as to the introduction of appropriate processes and boards. After reading this book, you should have a deeper understanding of SOAs, enabling you to effectively argue the benefits to different stakeholders and to establish the necessary processes and boards to make your SOA endeavor a success!

If you are a **software designer**, **analyst**, or **developer** working in an SOA project, although you are likely to work in a specific part of your application landscape, this book will help you obtain a better understanding of the entire process. Furthermore, there are key challenges such as process integrity that directly impact your work. This bookin particular <u>Chapters 7</u> to <u>10</u>helps to address these challenges in a coordinated manner within your SOA project.

If you work in the **IT strategy** department of an large organization, you should read this book in order to find out how SOAs can add to your IT strategy. Your work is likely to be driven by the demand for agility and cost effectiveness. Many enterprises have experienced projects that failed to deliver the required functionality and therefore lost business opportunities. Furthermore, many application landscapes suffer from high maintenance costs for their inherited assets and the integration of new applications. In <u>Part II (Chapters 1113)</u> you will read about the various possibilities for overcoming these issues with an SOA. Finally, several strategies for introducing the SOA within the organization are presented. <u>Part III (Chapters 14 to 17)</u> contains several case studies with real-world evidence that validates the SOA approach. Those success stories provide "living proof" of SOA success and offer an impression of the different ways an SOA can be established.

If you are an experienced **project manager**, you should read this book in order to understand the specific benefits of SOAs for project management. The SOA approach implies a major simplification of the overall software development process, and this book makes these benefits accessible. However, SOAs will challenge you, and as a result, this book presents solutions to the most important problems one encounters in an SOA project, both from the technical and project management viewpoints. You will find <u>Chapter 13</u>, which focuses on project management, and <u>Chapters 11</u> and <u>12</u>, which depict the political environment, to be most beneficial. It should be noted that this book does not introduce a new software development methodology. You will require a sound knowledge of your

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## A Roadmap for This Book

The successful adoption of an Enterprise SOA is based on three fundamental factors: architecture, organization, and lessons drawn from real-world experience. The IT architecture is the technical enabler for an SOA. A successful SOA adoption accelerates an enterprise by reducing the gap between strategy and process changes on one hand and supporting IT systems on the other. The IT architecture and the business organization are mutually dependent, although they both drive each other. Finally, real-world experience, in particular previous long-term IT infrastructure initiatives (both successful and unsuccessful) influence and validate many of the core concepts of SOA. Not surprisingly, this book is structured around these three factors. After we introduce the subject area in Chapters 1 to 3, Part I, Chapters 4 to 10, focuses on the **architecture**. Part II, Chapters 11 to 13, discusses the challenges of introducing an SOA at the level of the **organization**, depicting its benefits, processes, and project management. Part III, Chapters 14 to 17, provides **real-life examples** of successful SOA introductions.

**Chapter 1**, "An Enterprise IT Renovation Roadmap," identifies the need for agility and cost savings as the main drivers for the introduction of SOAs.

**Chapter 2**, "The Evolution of the Service Concept," describes how commercial information technology has moved toward the service concept over the last 40 years. Today's SOA is the preliminary endpoint of many years of painful "testing." Knowing and understanding previous pitfalls and mistakes help to avoid them in new projects.

**Chapter 3**, "Inventory of Distributed Computing Concepts," introduces the fundamental concepts of distributed computing that are required for subsequent discussions in <u>Part I</u> (<u>Chapters 410</u>). Particular topics will be communication infrastructures, synchronous versus asynchronous communication, payload semantics, granularity, and loose versus tight coupling.

## Part I: Architectural Roadmap

**Chapter 4**, "Service-Oriented Architectures," describes the particular requirements of large organizations for building an architecture and defines the term "Service-Oriented Architecture" as it is used throughout this book.

**Chapter 5**, "Services as Building Blocks," is a direct continuation of <u>Chapter 4</u>. It introduces different service typesnamely basic, intermediary, process-centric, and external servicesand gives an in-depth discussion of their key characteristics.

**Chapter 6**, "The Architectural Roadmap," completes the discussion started in <u>Chapter 5</u>. Using the concept of building blocks, the high-level structure of SOAs is depicted. <u>Chapter 6</u> introduces two key concepts: SOA layers and expansion stages. SOA layers aim to organize the aforementioned services at the enterprise level. Expansion stages are well-defined levels of maturity of an SOA that enable a stepwise implementation. In this book, three expansion stages are distinguished: fundamental SOA, networked SOA, and process-enabled SOA.

**Chapter 7**, "SOA and Business Process Management," shows how SOAs and BPM can complement each other in practice. This chapter draws a demarcation line between the responsibilities of a BPM infrastructure and the functional infrastructure provided by the SOA.

**Chapter 8**, "Process Integrity," delves into the challenges of distributed architectures with respect to consistency and how SOAs approach this major issue. This chapter provides numerous helpful, hands-on guidelines tackling real-world constraints such as heterogeneity, changing requirements, or budget.

**Chapter 9**, "Infrastructure of a Service Bus." By this point, the reader will know a lot about service types, the handling of business processes, and SOA layers. This chapter will address the issue of the type of runtime infrastructure that is required in order to put an SOA in placean infrastructure that is commonly known as the "service bus." <u>Chapter 9</u> highlights the fact that the service bus is often heterogeneous and provides technical services such as data transport, logging, and security.

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

## Chapter 1. An Enterprise IT Renovation Roadmap

This book makes a big promise: It is offering an *IT renovation roadmap*, which will leverage the concepts of Service-Oriented Architectures (SOA) on both the technical and organizational levels in order to create sustainable improvements in IT efficiency and agility. The aim of this roadmap is to strike a good balance between immediate gains on one hand and long-lasting improvements to the enterprise IT landscape on the other. An SOA should increase the capability of an enterprise to address new business requirements on the short term by reusing existing business logic and data models, thus incurring only minimal cost, resource, and time overheads, while minimizing risks, especially when compared to rewriting entire application systems. In addition, an SOA should provide endurable benefits in terms of agility because it provides a long-term strategy for the increase of the flexibility of an IT infrastructure.

This chapter closely looks at the problems faced by enterprise software today, the resulting requirements for an enterprise IT architecture such as an SOA, and how such an architecture can be established on the organizational level.

Team LiB



Team LiB

## 1.1. Agony Versus Agility

In 2003, Nicolas G. Carr published the heatedly debated article "*IT doesn't matter*" in the *Harvard Business Review*, claiming that "*... like electrical grids or railroads, IT would become a ubiquitous commodity*." Regardless of your position on this issuewhether or not you consider enterprise IT a commodityenterprises heavily depend on the IT backbone, which is responsible for running almost all processes of modern enterprises, be they related to manufacturing, distribution, sales, customer management, accounting, or any other type of business process. Because of today's highly competitive global economy, these business processes underlie constant change: Enterprises must constantly sense changes in market conditions and swiftly adapt their strategies to reflect these changes. Therefore, it is a key requirement for modern enterprise IT that changes in company strategy be reflected quickly and efficiently in the company's IT systems, which are the backbone for executing the strategy.

This is exactly where the enterprise software dilemma starts: Today's enterprise software development almost always suffers from *lack of agility* and from *inefficiency*. This means that enterprises are not able to match business requirements onto underlying IT infrastructure fast enough, effectively limiting the capability of the enterprise to react appropriately to market demands. In addition, the inefficiency of enterprise software development means that the development that is actually done costs too much when compared to the actual output.

If we look at a typical enterprise software system, we can normally observe an initial phase of high productivity and agility, as shown in Figure 1-1. During this Green field phase, the system is built with much new functionality, and initial change requests can be implemented relatively quickly and efficiently. However, after the initial system implementation has been put in place and the first couple of change requests have been executed, the ability to make more changes to the system deteriorates dramatically, and maintenance over time becomes harder and harder.

## Figure 1-1. Change requests reduce the agility of a system over time.

This stagnation phase, which almost any enterprise software system experiences over time, cannot be explained by a single reasona number of factors contribute to this phenomenon. Some of these reasons are related to software technology, such as the difficulty of making structural changes to an existing code base. However, most of the reasons are not of a technical nature but rather are related to reasons on the organizational level. For example, after the initial launch phase of a system, the project management and key domain experts are likely to move on to the next project, often leaving the maintenance of the system to less skilled maintenance workers, many times even without doing a proper hand-over. In addition, after the initial phase of euphoria about the new system, it might lose its internal lobby over time and thus the necessary political support within the organization. Tight project budgets often mean that fixes and changes cannot be done properly and are only done on an ad-hoc basis, without taking the existing order of the system into consideration. Typically, there is no time and budget for doing a proper refactoring of the system to ensure its long-term maintainability. Finally, critical structural decisions are often made on the side, without executing proper controlfor example, an engineer might quickly write a small batch program to synchronize data between two systems, and 10 years later, a small army of developers is needed to deal with the consequences of this ad-hoc decision.

When looking at enterprise software, we are usually not looking at isolated systems, but at large numbers of systems with complex cross-dependencies that have grown over many years and with a high level of heterogeneity and redundancies. We rarely have sufficient in-house knowledge about the different systems, and we often need to cope with very different design and programming styles. Finally, people are getting used to this situation and are starting to think in terms of workarounds, not in terms of the "right" structures.

Some organizations might have found better ways of coping with these problems than others, but it's hard to find any organization today that can claim that it has completely sorted out these issues. For this reason, many organizations require an *enterprise IT* repovation roadman to belo providing a sustainable transformation into a more agile IT.

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

## **1.2. Enterprise Software Is a Different Animal**

In order to better understand the problems of enterprise software, we need to look at the specific characteristics of it, which are different from those of other types of software, such as system software, desktop applications, embedded systems, scientific software, or video games.

As the name indicates, *enterprise software is tightly coupled with the internal organization, processes, and business model of the enterprise*. Enterprise software underlies both cross-departmental dependencies and external business relationships. Consequently, an architecture for enterprise software must deal with large numbers of different requirements. Many of these requirements are conflicting, while others are unclear. In almost every case, the requirements are a moving target due to the permanent change of markets, the organization of the enterprise, and its business objectives. It is this involvement in all aspects of the enterprise and the business that makes enterprise software highly complex.

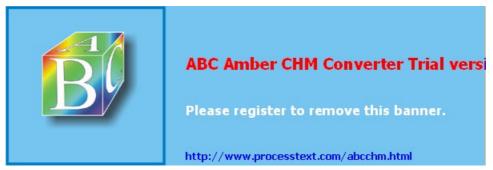
Enterprise applications rarely contain a large amount of complicated algorithms. The code that describes a piece of business logic is usually very simple. The structure of a COBOL-based billing system is much simpler than, for example, an embedded system for a Mars robot with complex real-time and multi-threading requirements. In enterprise applications, one usually finds very simple data structures, which again are different from other systems such as geographic information systems (GIS).

Let's consider an example in order to illustrate the difference between enterprise applications and other software: An enterprise application such as a Customer Relationship Management System (CRM), a billing system, a shipping system, or an insurance claims processing system. The stakeholders in these systems include different business units and potentially even the CEO, as well as different IT projects, IT maintenance, and operations. In these scenarios, we will be facing highly heterogeneous teams and often very political environments. The technology landscape will be highly heterogeneous as well, including many different application and middleware platforms. The business data and content will have a very long lifetime, especially when compared with the much shorter cycles of technology innovation. We need to deal with constantly changing functional requirements that are usually not well-defined. In addition, we will be facing many cross-dependencies between functional requirements, as well as heterogeneous technology platforms. The number of end users will be potentially very large, and the applications will have to be rolled out to large numbers of PCs, often more than 10,000.

Take, on the other hand, a desktop application, such as a word processor or spreadsheet application. A smaller, more homogeneous technical team will develop this application. It will be used by office workers as well, but the problem space is more well-defined. The application logic is self-contained, with very few cross-dependencies. Finally, there is no roll-out problem because the end user is typically responsible for the installation himself.

As we can see from these examples, enterprise software is unique in many respects, and therefore, it requires unique measures to ensure the efficiency of its development and maintenance.

#### Team LiB



# **1.3. The Importance of Enterprise Software Architectures**

According to the second law of thermodynamics, *any closed system cannot increase its internal order by itself*. In fact, any activity that is geared toward ordering the system will increase its overall disorder (called *entropy*). In many respects, this law is also applicable to enterprise software, which often has very similar characteristics. Consequently, *outside intervention* is continually required to help create a higher order and to ensure that development efforts are not lost.

In enterprise software, the architect takes on the role as an outside influencer and controller. It is his responsibility to oversee individual software projects from the strategic point of view of the overall organization, as well as from the tactical, goal-oriented viewpoint of the individual project. He has to balance different requirements while attempting to create an enduring order within the enterprise software landscape. The enterprise software architecture is the architect's most important tool at hand. Software architects are constantly confronted with changes to and expansion of functionality that increase system complexity and reduce efficiency. By refactoring current solutions, architects constantly strive to reduce complexity and thereby increase the agility of the system (see Figure 1-2).

## Figure 1-2. Software architects use refactoring to fight the constant increase in system complexity.

[View full size image]

Apart from the events that increase complexity during normal usage of the architecture, single events can also have an important effect on enterprise IT. They might occur in major changes to existing jurisdiction, the end-of-life of a supported product, or the introduction of large chunks of computing infrastructure, such as in the course of a merger or acquisition. Such events require a major effort at very short notice to keep the architecture in a simple and maintainable state. Devastating consequences have been observed as a result of mergers and acquisitions: concise financial reporting being lost after a merger and raw system capacity being exhausted after an acquisition. Because it is unknown *a priori* when such effects will occur, it is vital to keep the enterprise architecture in a maintainable and changeable state all the time.

As we will see in the remainder of this book, Service-Oriented Architectures are particular well suited to cope with the needs of such an ongoing incremental process of optimization.

Team LiB



Team LiB

# **1.4. The Requirements for an Enterprise Software Architecture**

As a result of the aforementioned tight coupling with the internal organization, processes, and business model of the enterprise, an enterprise software architecture must fulfill very different requirements than, for example, a software architecture for a system that is controlled by a small number of highly qualified domain experts, such as the Mars robot or a video game engine.

In order to improve agility and efficiency, an enterprise software architecture must provide particular characteristics:

**Simplicity.** The enterprise architecture must be simple in order to allow efficient communication between key personnel. As previously discussed, many people are involved in the specification and construction of enterprise software. All these people have different roles and consequently different viewpoints with regard to the software. It is also likely that several different skill sets exist among personnel. These might range from IT coordinators of functional departments to technical architects. Many IT coordinators will have detailed business domain knowledge but no technical expertise. On the other hand, technical architects will probably have an excellent technical education but have little understanding of the vertical business. Nevertheless, all the people involved must be able to understand and manage the architecture at their respective levels (e.g., specifying new functionality at the business level and implementing and maintaining it).

**Flexibility and maintainability.** Every enterprise system is subject to ongoing change. It must continuously be adapted to new requirements due to the need of evolving markets, legal changes, or business reorganizations. Therefore, the architecture must lead to a highly flexible and maintainable system. The architecture must define distinct components that can be rearranged and reconfigured in a flexible manner. Local changes cannot be permitted to have an impact on the global system. Providing that the external API of a component remains stable, an internal change should not affect operations outside the components must be designed very carefully. To a great extent, interfaces must be generic and not specific to a single usage scenario. However, defining generic interfaces requires excellent domain knowledge, experience, and to some extent, luck. Finally, the internal implementation of a component must allow efficient maintenance, making it is easy to add or modify functionality.

**Reusability.** Reusability has been a major objective of software engineering for decades, with varying degrees of success. It is in the interest of an enterprise to gain as much benefit from its software assets as possible. This can be achieved by creating an inventory of useful building blocks and continually reusing them. One obvious reason for reuse is reduced development and maintenance cost, which can be accomplished by sharing common functionality in code libraries that are used across different projects. However, perhaps a more important aspect of reusability is the ability to share data across applications in real-time, thus reducing content redundancies. Having to maintain the same dataset in multiple databases becomes a nightmare in the long term. Unfortunately, it is not easy to achieve the goals of reuse. Large organizations have learned that reuse is not always efficient because it is particularly costly to administer, find, and understand the components that should be reused, and sometimes this cost outweighs the benefits.

**Decoupling of functionality and technology.** The architecture must make an enterprise organization independent of the technology. It must decouple the long lifecycle of the business application landscape from the shorter innovation cycles of the underlying technology. Moreover, an architecture that is designed to last longer than one or two of these technology innovation cycles must cope not only with changing technologies but also with the actual lifecycles of installed technologies, which can be much longer. It is therefore a major requirement that the architecture tolerate both heterogeneity and change to its technical infrastructure. Furthermore, the development of business functionality must be decoupled from the underlying technology. In particular, the architecture must avoid dependencies on specific products and vendors.

This book illustrates how a Service-Oriented Architecture can help achieve the design goals for enterprise software systems as described previously.

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## **1.5. The Relation of Enterprise Architecture and Enterprise Standards**

For many decades, enterprise IT organizations have attempted to improve agility and efficiency by homogenizing their systems through the introduction of enterprise-wide IT standards, but mostly with very limited success. Therefore, it is important to understand that an enterprise architecture is not equal to an enterprise standard, as we discuss in this section.

In the 1980s, with relational database systems becoming mainstream, we saw a wave of so-called *Enterprise Data Model* (EDM) projects. The idea of these standardization projects was to define one global data model for all the business entities in an enterprise, which was to be shared among all the different organizations and systems in a company. Almost all of these EDM projects failed, and today, there are usually as many different database schemas out there as there are databases in an enterprise. There are a variety of different reasons for the failure of these EDM projects, including political turf wars between different departments, conflicting interests between the different stakeholders ranging from business representatives over application specialists to DBMS administrators, the sheer technical complexity of the undertaking, and the fact that due to the dynamics and complexity of modern enterprises, it is usually impossible to capture a snapshot of the complete state of the enterprise at a given point in time.

In the 1990s, we saw the next attempt to homogenize the enterprise application landscape, this time through enterprise-wide middleware standards. The concept of the *Enterprise Software Bus* became popular. The idea was that by agreeing on a ubiquitous, technology-independent, enterprise-wide standard for communication between software modules, the problem of application integration would be solved once and for all. However, the reality in almost all enterprises today is that in addition to application heterogeneity, we now face the problem of middleware heterogeneity as well. In many cases, middleware such as CORBA was only used to solve point-to-point integration problems on a per-project basis, instead of being established as a global software bus; as a result, many enterprises now have nearly as many incompatible middleware systems as they have applications.

In general, it seems fair to say that enterprise standardization efforts in IT have failed to deliver on their promise of homogenization and easy application integration. Too many generations of middlewareranging from DCE over CORBA to SOAP and WSDLhave been touted as silver bullets but have failed to become established as *the* ubiquitous Enterprise Software Bus, leaving behind a high level of cynicism among the people involved.

As a reader of a book on Service-Oriented Architectures, you might now be asking yourself, "So what is different this time?" Is SOA not yet another enterprise-wide standardization effort, this time under the label of the *Enterprise Software Bus*? How are SOAP and WSDLwhile maybe technically superior and more flexiblegoing to address the organizational challenges of global standards that made the Enterprise Data Model, the Enterprise Software Bus, and many other enterprise standardization efforts fail to a large extent?

This book takes the position that SOA is neither a technology nor a technology standard, but instead it represents a technology-independent, high-level concept that provides architectural blueprints, such as the ones outlined in the first part of this book. These architectural blueprints are focusing on the slicing, dicing, and composition of the enterprise application layer in a way that the components that are created and exposed as services in the SOA are not only technically independent but also have a direct relationship to business functionality. They enable the structuring of application components on the local level while also catering for global integration of these components. As we will show in this book, an SOA does not rely on the support of particular runtime protocols, such as SOAP or IIOP. Therefore, an SOA does not impose adherence to technical standards on the global level and is not based on strict norms and specifications (see Figure 1-3).

#### Figure 1-3. Enterprise Data Models and Software Buses were popular approaches to the challenges of enterprise computing in the 1980s and 1990s.

[View full size image]

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

## **1.6. Organizational Aspects**

When talking about enterprise IT, it is important to realize that manyif not mostof the problems associated with it are not of a technical nature but can be found on the organizational level instead. Quite naturally, we have already implicitly touched on many of these organizational aspects in our discussion so far (for example, when discussing the reasons for the failure of enterprise standards such as the *Enterprise Data Model* or the *Enterprise Software Bus*, which largely resulted from problems on the organizational and not the technical level).

The IT organization and the way projects are managed in a large enterprise are again very different from what one would find, for example, in a company that produced embedded systems or games. First and foremost, it is important to realize that most likely in no other part of the software industry will we find a development and maintenance process that is so closely aligned with the end customer. If an enterprise is developing a new financial reporting system, it will have to be done hand-in-hand with the finance department and any other stakeholders of the financial reporting system, possibly up to the CEO. A software team that is developing embedded control software for a dishwasher is unlikely to have daily meetings with a housewife about the exact functionality of the software.

An important consequence is that we are dealing with a much more complex and more ambiguously defined decision-making process, which is driven more often by business strategy and political agendas than by technical arguments. The organizational environment we are dealing with is extremely heterogeneous, and many different opinions will have to be incorporated into any decision that is made, be it a decision about budgets, functional requirements, project priorities, or the interesting question of what actually defines the success of an IT project.

For all these reasons, it is vital that our *enterprise IT renovation roadmap* provides not only a *technical roadmap* but also an *organizational roadmap*, which outlines how the technical architecture is to be established on the enterprise level from the political and organizational point of view. The second part of this book provides an overview of this organizational roadmap.

#### Team LiB



#### Team LiB

## 1.7. Lifelong Learning

Enterprise software has always suffered from the mismatch between technical and business-related concepts and the different languages spoken by the people on both sides of the fence. As a result, we have not only faced inefficiencies, but we also have often lost important knowledge and consequently had to reinvent many solutions.

Many attempts have been made in the past to find a common denominator between business and technical concepts. For example, SQL was invented in the 1970s with the vision that it would give non-technical business analysts a tool to access, analyze, and manipulate business data directly. Today, SQL is largely seen as a tool for technical experts, and it has turned out that most of the entities found in relational databases are too fine-grained and closely intertwined with technical concepts to have a meaning on the business level.

It is a key goal of an SOA to provide services that have a concrete meaning on the business level. Because of this one-to-one mapping between business and technology entities, SOA provides a unique chance for the first time in IT history to create artifacts that have an enduring value for both the business as well as the technology side. SOA provides a chance to make things that have been learned the hard way usable for the organization in the long run.

Similarly to human beings, organizations will never be able to stop learning if they want to be successful for long. SOA provides an excellent platform for this lifelong learning on the organizational level because an SOA enables us to constantly compare the nominal and the actual and to react accordingly to fill the gaps or adapt the architecture to reflect changes in business strategy.

Consequently, the third part of this book provides a number of real-world case studies, which can provide a good starting point for learning the lessons resulting from other organizations' adoption of SOAs and the impact they had.

Team LiB



## **1.8. The Enterprise IT Renovation Roadmap**

As we have outlined in this introduction, we need strong enterprise architecture concepts to address the structural problems of the enterprise IT landscape, accompanied by a strategy for how to establish the architecture on the organizational level. SOA provides these concepts, but we have to be aware that implementing an architecture like an SOA is an ongoing process, requiring constant guidance and overseeing. The SOA architect needs to bridge many conflicting requirements, resulting from frequent changes of business requirements, the evolution of application and infrastructure technology, and last but not least, changes to the architecture itself. We need ways to introduce step-wise improvements, which will bring us slowly but steadily closer to our goal. We will have to accept that the path we are about to enter will not be a straight path, and there will be many influences outside of our control. Nevertheless, the authors believe that the introduction of an SOA will bring many long-term benefits to an enterprise. Figure 1-4 depicts the overall vision for how our roadmap can bring an enterprise that is suffering from development inefficiencies back to a level of high efficiency and agility.

#### Figure 1-4. Service-Oriented Architecture is a key element of an enterprise IT renovation roadmap.

[View full size image]

This book aims to flesh out an *enterprise IT renovation roadmap*. This roadmap is not only related to technology, but it also equally addresses organizational challenges. The roadmap anticipates constant changes in strategic directions and assumes that the underlying architecture will have to be constantly adapted to include the results of lessons learned as we go along. Consequently, the three parts of this book are structured to capture these different requirements.

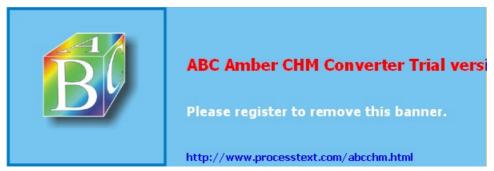
#### Figure 1-5. An Enterprise IT renovation roadmap needs to address three dimensions: architecture, organization, and real-world experience.

<u>Part I</u> of this book provides the *architectural roadmap*, mapping out the different *expansion stages* of an SOA, which will gradually help IT organizations to get back to a high level of agility and efficiency on the technical level.

Part II of this book looks at the *organizational roadmap*, providing an in-depth discussion of how the architecture can be established on the organizational level, the different stakeholders and how they must be involved, and how an SOA can be leveraged to drive project management more efficiently.

Finally, <u>Part III</u> of this book provides a number of case studies, which provide *real-world experiences* from large corporations that have already started their journey on the SOA-based enterprise renovation roadmap.

Team LiB



Team LiB

## Chapter 2. Evolution of the Service Concept

Before looking at the road ahead, we want to take a step back and look at the evolution of the service concept by examining the milestones of enterprise computing and how they have shaped the concept of "services." We will look at three core development directions: programming languages, distribution technology, and business computing. Each has undergone a major evolution in the past 40 years, leading to a level of abstraction that supported the emergence of Service-Oriented Architectures.

#### Team LiB



Team LiB

## 2.1. Milestones of Enterprise Computing

The term "service" has been present in commercial computing for a long time and has been used in many different ways. Today, for example, we find large companies, such as IBM, promoting the concept of "services on demand." At the beginning of the new century, the term "Web services" became extremely popular, although it has often been used to refer to very different computing concepts. Some people use it to refer to application services delivered to human users over the Web, like in the popular salesforce.com application. Other people use the term "Web services" to refer to application modules made accessible to other applications over the Internet through XML-based protocols.

Because of the many different ways in which the term "service" has been used over the years in the IT industry, it is necessary to define more precisely how we use it in this book. However, before looking at a more formal, technology-oriented definition in <u>Chapter 4</u>, "Service-Oriented Architectures," we will look at a more generic definition that better suits the purpose of this chapter, which is examining the roots of "our" understanding of services.

The Merriam Webster's Dictionary gives various definitions for the term "service," including *"useful labor that does not produce a tangible commodity"* and *"a facility supplying some public demand."* In this book, the term "service" takes its meaning from these definitions. It denotes some meaningful activity that a computer program performs on request for another computer program. Or, in more technical terms, a service is a remotely accessible, self-contained application module. Application frontends are making these services accessible to human users (see Figure 2-1). Often, the terms "client" and "server" are used synonymously for "service consumer" and "service provider," respectively.

#### n http://www.m-w.com.

#### Figure 2-1. Our understanding of the term service: A service provider (commonly a remote server) performs some task at the request of a service consumer (the client).

The services covered in this book provide abstraction from a lot of their technical details, including location and discovery. Typically, our services provide business functionality, as opposed to technical functionality. Consistent with the definition from Merriam Webster's Dictionary, our services are not designed for one specific customer, but instead they are "a facility supplying some public demand" they provide a functionality that is reusable in different applications. The cost effectiveness will depend strongly on the number of different customers a service has, that is, the level of reuse that can be achieved.

A concrete implementation of an SOA provides service consumers with seamless access to the different services available in it. Thus, after a service consumer is "wired" into the instance of the SOAafter the "ring tone" of the SOA is availableusage of the services is seamless and transparent. However, <u>Chapter 1</u> said that an SOA is an architecture *per se* and not a service bus or any other specific middleware, and therefore, it describes structure, not concrete technology. Consequently, instances of SOAs might take very different technical shapes and forms in different enterprises.

A crucial factor in the development of services as we understand them in the context of this book is the quest for the right degree of abstraction. Ultimately, a service encapsulates some activity of a certain complexity. Using a service makes the world more convenient for the service consumer. Consequently, an appropriate interaction pattern must exist between the service provider and service consumer. Given the analogy of the telephone network as the service infrastructure, services can be anything from hectic chatter to concise and focused conversation. They can also include some form of telephone conference, answering machine, or call redirection. In the remainder of this book, you will notice surprising similarities between the service concept and the telephone analogy.

It is interesting to notice that the service model, as it is defined in this book, has been preceded by many technologies and technical concepts in the last 30 years that shared many of the same underlying ideas and concepts. For example, look at the creation of

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 2.2. Programming Paradigms

The first programming paradigm that enabled abstraction from the details of computer programs was functional decomposition and the related technology of functional analysis. Functional decomposition pioneered the formal introduction of flow charts, showing data that flows through a number of processes. One of the first people to formalize it was Myers in 1976 [Myers76]. However, the first language that was suited to functional decomposition was the COBOL (Common Business Oriented Language) programming language that was created as early as 1959 by a body called CODASYL (Conference on Data Systems Languages) and that was later adopted as an ANSI standard. Having undergone many improvements and additions since then, it is still a prevailing programming language in many major organizations. Apparently, more code is written in COBOL than any other language. In 1970, while working at the Polytechnic University of Zurich, Niklaus Wirth invented Pascal, a language that explicitly encouraged functional decomposition and that remains one of the most popular teaching languages for computer science.

Functional programming concepts remain popular because they are easy to understand by students, programmers, and customers. They provide a powerful tool to create reusable blocks of codefunctionsthat can even be sold in the form of software libraries.

Functional programming contributed to the service concept because functions essentially provide some form of abstraction. However, the amount of abstraction they can provide is limited.

It soon became apparent that the functional paradigm had its limits. Multipurpose reusable functions are hard to create. Often, the caller must provide many parameters, and a lot of data must be passed into multiple functions in order to obtain the required result. The concepts of software modules and software components were created to cope with this growing complexity. These concepts came in many different flavors. The first time these concepts appeared was in the original implementation of the ADA programming language, modules in the Modula2 computing environment (also created by Niklaus Wirth in 1979), and the hugely commercially successful MS Visual Basic's VBX components. Although very different, they share the common abstraction of software components as a container for both data and the functions that operate on that data. Even before the advent of software components, it was considered good programming practice to shield a function's user from its internal details. At this point, the concept was introduced as a language element known as encapsulation.

The significant increase of abstraction and encapsulation that components provide are an important step towards service orientation. However, their main purpose was in-situ development reuse, while service orientation focused on distribution and runtime reuse.

By the early 1980s, modularization and component programming were widely recognized as the next big trend in software development, and the MODULA language provided a stable and mature programming platform for this trend. However, the Japanese and U.S. governments poured massive amounts of money into the development and marketing of their own programming environments, PROLOG and ADA.

The uncertainty that emerged from this proliferation of platforms delayed the adoption of component-oriented programming long enough for it to be outrun in popularity by object-oriented programming, which introduced the object as a programming and runtime concept. Originally, object-oriented programming was developed for simulation purposes. The first object-oriented language, SIMULA, was developed as early as 1967 at the Norwegian Computing Center in Oslo by Ole-Johan Dahl and Kristen Nygaard, and it has since been developed further [Kirk89]. Object orientation entered mainstream programming paradigms in the mid 1980s with the creation of Smalltalk [Gold83] and C++ [Stro85]. New versions of most other object-oriented languages, such as Java, were invented, while others, such as Pascal or even COBOL, were extended to embrace object orientation in one way or another.

It took approximately 15 years until the concepts of component-orientation reemerged in the late 1990s, this time supported by concrete component platform implementations such as Enterprise Java Beans.

Objects are much like components in that they support encapsulation and the bundling of data and functions and add the concept of individual entities (the objects) as instances of classes. The objects communicate through message exchange, but more importantly, object orientation adds the concept of inheritance, where types can be derived from other

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 2.3. Distributed Computing

Where the term service is used in this book, we usually assume that the service does not necessarily reside on the same physical machine as the calling application. The capability to call a remote computer program from another computer program in a seamless and controlled way is at the heart of our understanding of services.

An important challenge of distributed computing is to find abstractions for both remoteness and the actual service task at the same time. Although remoteness can be hidden from the service consumer, the service provider must still choose its invocation scenarios with the right granularity.

The distributed computing infrastructure was developed in the last 30 years. Business computing originally meant mainframe computing and involved large computers with costs in the multimillions of dollars, performing tasks mostly on their own. At some point, they morphed into more interactive multi-user systems. Rather than distributing the computing power, only data capture and display was distributed using terminal devices such as the DEC VT100 or the IBM3270. Some of the first things such systems had to share among themselves were data and output devices such as tape recorders or printing systems. In the early 1970s, computers became smaller and cheaper. The price/performance ratio made computer technology suitable for a broad range of applications. Research institutions quickly realized that they could operate both more economically and more independently when they were able to use various small computers rather than one mainframe system. At the universities of Stanford and Berkeley, two research programs eventually led to the creation of the Unix operating system. The Stanford University Network spun off the company Sun Microsystems in 1982, which is today one of the largest vendors of Unix computers. Unix is different from its predecessorsand many of its successorsin that its design guickly adopted the network as an essential part of the system environment. Two ideas fostered this design perspective: facilitating remote control of computers and programs and providing services to other computers in the network. The first train of thought created tools such as *telnet* and the Berkeley *r-tools* suite. The second one featured remote printing and the transparent provision of storage spacethe NFS file system released by Sun Microsystems in 1984. In fact, the latter was the original raison d'etre for the SUN-RPC standard, the first Remote Procedure Call system.

Even though distributed computing was available as early as 1980, it was mainly confined to the academic world until well into the 1990s. Unix computers mainly acted as so-called workstationspowerful computational and visualizing enginesin research facilities and universities. In the business environment, a hub-and-spoke distribution model prevailed until well into the 1990s. A number of desktop computer systems would typically access a central system for storage and printing. Often, these file servers used an entirely different operating platform from its clients. As structured and relational databases became more mature, businesses adopted a client/server approach. A large chunk of the application resided with the client that remotely accessed a database server. Execution logic was split between client and server as databases, most notably Sybase, introduced the concept of functions that were executed within the database and that did not need to be shipped with the client applicationso-called stored procedures. Another remarkable innovation was Novell's NetWare Loadable Modules (NLM), which were programs that ran on the server.

The next logical step was to blur the distinction between the client and the server. Combining concepts from distributed computing platforms such as the Distributed Computing Environment (DCE) with the newly emerging paradigm of object-oriented development, CORBA (Common Object Request Broker Architecture) was created. Instead of providing servers, which expose large numbers of remotely accessible functions, the functionality was now broken down into uniquely identifiable, remotely accessible objects that were able to manage their own state. Different objects could communicate with each other by means of an Object Request Broker (ORB). To connect to these objects, no knowledge of where they actually reside is necessary. Instead, an ORB provides abstraction mechanisms, such as naming services, which take care of the runtime discovery of the objects. Similarly to object-oriented programming, CORBA embraced the concept of programming by interfacein fact, all CORBA objects can be implemented in various programming languages, while their interfaces are described using a common Interface Definition Language (IDL).

Technically very elegant and sophisticated, CORBA promoted the actual reuse of live

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 2.4. Business Computing

Although the evolution of programming languages and distributed computing eventually provided the technical concepts that today are the first cornerstone of Service-Oriented Architectures, it is equally important to look at the developments in business computing that provided the content that represents the second cornerstone of Service-Oriented Architectures: business data and business logic.

The history of computing was always closely related to solving business problems, starting as early as 1940 with computers being used as large-scale calculators and as a replacement for large filing cabinets. Functions that nowadays are considered "technical" provided immediate business value in the advent of computing.

The following decades created further levels of abstraction, making it easier to think of a computer in terms of a provider for business services. However, business computing maintained its focus on mainframe computers. Most software was custom-built, originally written in machine language and later in functional languages such as COBOL or FORTRAN. Yet business computing proved a crucial success factor for enterprises. For example, logistics companies used computers to compute routes for shipments through their vast international transport networks. Retail giant Wal-Mart was among the first to create custom-made supply-chain management systems to optimize the purchase and distribution of goods.

In the 1970s, the original homegrown filing cabinet applications were gradually replaced using fully fledged database systems, including relational databases, which encapsulate the storage of complex interrelated data. Although today we regard the storage mechanism itself as a technical concept, it seemed fairly business-oriented when it was created. In fact, SQL was developed as a language that was intended to be used mainly by business analysts, not database programmers.

In 1972, four former IBM employees founded SAP in Germany, an event that marked a milestone for business computing. SAP introduced R/2 in 1981, the first business-computing platform that enabled enterprise-wide real time processing of financial data and resource planning information.

However, by the mid 1980s, corporate software development seemingly reached saturation level. Most companies were convinced that they had achieved most of what was possible with computing in their environment. College students were discouraged from majoring in software development because people assumed that only maintenance would be needed in the future, which probably would be performed in some remote offshore location.

Then computers began to reach employee desktops. Systems such as the Commodore PET and others introduced a new concept to computing. Data was obtained from remote storage, while computation and visualization were performed locally. This paradigm was boosted by the huge success of the IBM PC, which was launched in 1984. Through several stages of hardware and software development at both the client and server sides, the client/server paradigm held steady. The focus for advancement shifted back and forth between the two. On one hand, client computers became more powerful, sporting graphical user interfaces and raw computational power, and networks became faster. On the other hand, database vendors worked hard to add value to their systems by providing fault tolerance, scalability, and load balancing using cluster techniques and procedures that were executed within the database.

Driven by an increasing economical globalization and new manufacturing models such as Just-in-Time production, the supply and distribution chains of companies became increasingly sophisticated, relying more on complex IT systems to manage these processes. The software market reacted to this newly awakened demand in enterprise computing by developing complex enterprise applications, such as Enterprise Resources Planning (ERP) and Supply Chain Management (SCM). Over two decades, the market for enterprise solutions has become increasingly complex, offering applications ranging from Customer Relationship Management and Product Lifecycle Management to highly specialized applications, such as applications that manage complex transportation networks or billing systems for telecommunications companies. As a result, a plethora of new enterprise software companies emerged, and old ones grew even bigger, including SAP, Siebel, Oracle, PeopleSoft, J.D. Edwards, Baan, Manugistics, and others.

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

## 2.5. Conclusion

This chapter looked at the historical developments of programming languages, distribution technology, and business computing and how each of these areas contributed to the development of service orientation.

The evolution of programming languages has not only provided us with more productive development platforms but has also significantly contributed to the understanding of interfacing techniques and access patterns for services in an SOA. A key lesson learned is that not all programming language concepts are suitable in distributed computing and that service orientation is a deliberate step back from object orientation, aiming to provide more coarse-grained components with simpler access patterns.

The evolution of distribution technology has given us a variety of remote access technologies from which we can choose today, together with an infrastructure for transaction management, security, load-balancing, failover, and other critical features.

Finally, the evolution of business computing has lead to the development of advanced enterprise applications, such as ERP and CRM, which are today providing the content that represents the second cornerstone of an SOAthe data and business logic that brings our services to life.

#### References

[Myers76] Myers, Glenford J. *Composite/Structured Design*. Van Nostrand Reinhold Co. 1976.

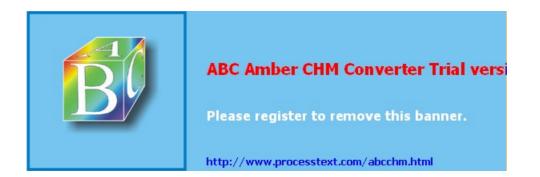
[Gold83] Goldberg, Adele and David Robson . *Smalltalk-80: The Language and its Implementation*. Addison-Wesley. 1983.

[Kirk89] Kirkerud, Bjorn . *Object-Oriented Programming with Simula*. Addison-Wesley. 1989.

[Stro85] Stroustrup, Bjarne . *The C++ Programming Language*. Addison-Wesley. 1991.

#### **URLs**

http://www.m-w.com Team LiB



## Chapter 3. Inventory of Distributed Computing Concepts

Before examining SOA elements in detail in the following chapters, we will review existing concepts of distributed computing. This is important because we are not planning to develop an SOA from scratch. Instead, an SOA will have to incorporate existing middleware technologies and distributed computing concepts. This is particularly important because earlier attempts to replace existing middleware with a new, ubiquitous software bus (e.g., CORBA) have failed, and a successful SOA will have to embrace existing and upcoming technologies instead of replacing or precluding them. Many authors cover the intrinsic details of communication networks and middleware, such as Tanenbaum [Tan2002, Tan2003] and Coulouris [Cou2001]. Aiming our discussion at the application architecture level, we will provide only a brief overview of the most fundamental communication middleware concepts here (including Remote Procedure Calls, Distributed Objects, and Message-Oriented Middleware), followed by a more detailed discussion on the impact that different types of communication middleware have on the application level (including synchrony, invocation semantics, and application coupling).

#### Team LiB



#### Team LiB

## 3.1. Heterogeneity of Communication Mechanisms

Techniques for the distribution of enterprise software components are manifold. As will be seen in the remainder of this book, this heterogeneity is inevitable due to the various communication requirements of enterprises.

The situation is comparable to communication in real lifemany forms of communication exist (verbal, non-verbal, written, etc.), and every form has its own purpose. It is not possible to replace one form with another without reducing expressiveness.

Figure 3-1 depicts three possible levels of heterogeneity of distribution techniques:

- Communication mode
- Products
- Additional runtime features

#### Figure 3-1. Distribution techniques for enterprise applications are characterized by manifold requirements and consequently by various dimensions of heterogeneity.

[View full size image]

*Communication modes* are basically distinguished between synchronous and asynchronous mechanisms. Evidently both are required in real-world projects. However, in practice, there are usually numerous variants of these basic modes of communication. Obviously, one can encounter numerous *products* that provide distribution mechanisms. In addition, a concept that is supposed to cover all the distribution issues of an enterprise must also provide a set of *additional runtime features* such as security support, fault tolerance, load balancing, transaction handling, logging, usage metering, and auditing.

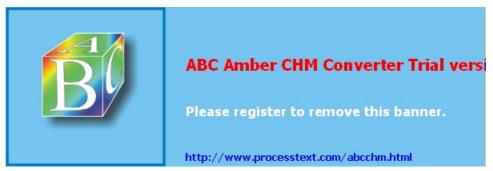
It should be noted that our classification scheme is arbitrary. It is possible to define other classifications or to find additional levels of heterogeneity. However, independent of the classification scheme, it is true that enterprise distribution techniques tend to create heterogeneity at different levels.

From a technical point of view, this scenario leads to three different layers, as shown in Figure 3-2. The first layer contains the core assets of the enterprise application landscape, including all business logic. The second layer provides technology-dependent adapters that connect the core assets to various software busses. Finally, the third layer represents the sum of the enterprise's communication facilities.

#### Figure 3-2. Technology-dependent adapters connect participants of an enterprise application landscape with its communication infrastructure.

The remainder of this chapter focuses on the second layer. <u>Chapters 4</u> to 7 provide an in-depth discussion of the first layer, while <u>Chapter 9</u> discusses the third layer.

#### Team LiĐ



Team LiB

## **3.2. Communication Middleware**

A communication middleware framework provides an environment that enables two applications to set up a conversation and exchange data. Typically, this exchange of data will involve the triggering of one or more transactions along the way. Figure 3-3 shows how this middleware framework acts as an intermediary between the application and the network protocol.

## Figure 3-3. A communication middleware framework isolates the application developers from the details of the network protocol.

In the very early days of distributed computing, the communication between two distributed programs was directly implemented based on the raw physical network protocol. Programmers were involved with acute details of the physical network. They had to create network packets, send and receive them, acknowledge transmissions, and handle errors. Therefore, a lot of effort was spent on these technical issues, and applications were dependent on a specific type of network. Higher-level protocols such as SNA, TCP/IP, and IPX provided APIs that helped reduce the implementation efforts and technology dependencies. They also provided abstraction and a more comfortable application development approach. These protocols enabled programmers to think less in terms of frames at OSI layer 2 or packets at layer 3 and more in terms of communication sessions or data streams. Although this was a significant simplification of the development of distributed applications, it was still a cumbersome and error-prone process. Programming at the protocol layer was still too low-level.

As the next evolutionary step, communication infrastructures encapsulated the technical complexity of such low-level communication mechanisms by insulating the application developer from the details of the technical base of the communication. A communication middleware framework enables you to access a remote application without knowledge of technical details such as operating systems, lower-level information of the network protocol, and the physical network address. A good middleware framework increases the flexibility, interoperability, portability, and maintainability of distributed applications. However, it is the experience of the recent two decades that the developer's awareness of the distribution is still crucial for the efficient implementation of a distributed software architecture. In the remainder of this chapter, we will briefly examine the most important communication middleware frameworks.

## 3.2.1. RPC

Remote Procedure Calls (RPCs) apply the concept of the local procedure call to distributed applications. A local function or procedure encapsulates a more or less complex piece of code and makes it reusable by enabling application developers to call it from other places in the code. Similarly, as shown in Figure 3-4, a remote procedure can be called like a normal procedure, with the exception that the call is routed through the network to another application, where it is executed, and the result is then returned to the caller. The syntax and semantics of a remote call remain the same whether or not the client and server are located on the same system. Most RPC implementations are based on a synchronous, request-reply protocol, which involves blocking the client until the server replies to a request.

#### Figure 3-4. RPC stubs and libraries enable location transparency, encapsulate the functional code for the RPC communication infrastructure, and provide a procedure call interface.

The development of the RPC concept was driven by Sun Microsystems in the mid 1980s and is specified as RFC protocols 1050, 1057, and 1831. A communication infrastructure with these characteristics is called RPC-style, even if its implementation is not based on the appropriate RFCs.

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 3.3. Synchrony

Synchronous and asynchronous communications are two different forms of interaction that both require the support of a generic technology for distributed systems.

Synchronous communication is characterized by the immediate responses of the communication partners. The communication follows a request/reply pattern that enables the free flow of conversation, often based on the use of busy waits. Applications with user interaction specifically require this conversational mode of interaction. Synchronous communication requires that the client and server are always available and functioning.

Asynchronous communication is less stringent. Both communication partners are largely decoupled from each other, with no strict request/reply pattern. Typically, one party creates a message that is delivered to the recipient by some mediator, and no immediate response is needed. The sender can store context information and retrieve it when the recipient returns the call, but there is not necessarily a response. In contrast to a synchronous request-reply mechanism, asynchronous communication does not require the server to be always available, so this type can be used to facilitate high-performance message-based systems.

Typically, synchronous communication is implemented by RPC-style communication infrastructures, while asynchronous mechanisms are implemented by MOM. However, it is entirely possible to implement synchronous communication based on MOM, and it is also possible to build MOM-style interaction over RPC. Nevertheless, RPC is more suitable if immediate responses are required, and MOM is the technology of choice for decoupled, asynchronous communication.

Due to the manifold requirements of most real-world scenarios, typical enterprise systems embody both synchronous and asynchronous communication. For this purpose, a variety of different communication infrastructures is used, ranging from simple FTP (File Transfer Protocol) to more advanced middleware platforms, such as RPC and MOM. In addition, there are also communication infrastructures that support both communication modesfor example, pipelining RPC, which supports asynchronous communication in addition to the standard synchronous RPC communication.

To conclude this discussion on synchrony, we will provide an overview of the most common ways of implementing synchronous and asynchronous communication with both RPC/ORB and MOM technology. We will look at the following examples:

- Simulated synchronous services with queues
- Asynchronous one-way: fire-and-forget RPC
- Callbacks and polling services

The first example, *simulated synchronous communication*, can often be found in mainframe environments, where a message queuing system has been chosen as the standard means for remote interaction with the host, such as OS/390 with MQSeries access. This is a common scenario in many large enterprises. Often, these companies have gone one step further, developing frameworks on top of this combination of OS/390 and MQSeries that enable service-like interfaces to the most widely used transactions on the mainframe. This is done by implementing client-service wrappers that shield the underlying MQ infrastructure from the client developer. These service wrappers often simulate synchronous interactions with the mainframe by combining two underlying queues, one with request semantics and the other with reply semantics, using correlation IDs to pair messages into request/reply tuples. Effectively, this relegates the message queuing system to playing a low-level transport function only, not generally leveraging any of the advanced features of the messaging system. Figure 3-9 provides an overview of this approach.

## Figure 3-9. Simulated synchronous services with queues. A correlation ID maps a reply message to the corresponding request. On the client side, this is hidden by a service wrapper, which gives the caller the impression of synchrony.

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## **3.4. Interface Versus Payload Semantics**

Typically, an interaction between a client and a server (or a sender and a receiver) results in the execution of a transaction (or some other activity) on the receiving end. In order to determine the type of transaction or activity that was requested by the caller (or sender), it is necessary to specify the operation. This is normally performed in one of two ways: The requested transaction/activity can be encoded in the operation signature of the server component's interface, or it can be embedded in the message itself.

In the first case, the requested transaction (or other kind of activity) is defined by using self-descriptive function names such as saveCustomer(), retrieveCustomer(), or transferMoney(). RPC-style interfaces provide this type of semantically rich interface, which we refer to as *interface semantics* (see Figure 3-13).

## Figure 3-13. RPC-style interaction is typically based on interface semantics. Every procedure call has a meaningful name that indicates its purpose.

In the second case, the requested transaction is embedded directly into the message (see Figure 3-14). This can be done as part of the message header (if the MOM provides such a field as part of the message header data structure), or as part of the application specific payload. We refer to this as *payload semantics*.

# Figure 3-14. The name of a remote call with payload semantics has no functional meaning in its own right. The remote functionality required is encoded in the message that is sent. The receiver typically has to determine the function name and the dispatch message to the associated business function.

Payload semantics is widely used in the context of MOMs that provide APIs with functions such as MQGET()/MQPUT() or sendMessage()/onMessage()/receiveMessage() for the clients and servers to communicate with each other. The semantics of these functions is purely technical (see Figure 3-15).

#### Figure 3-15. MOM is generally based on payload semantics. Functions such as sendMessage() and processMessage() are purely technical, without any business semantics.

Interface semantics provide users with well-defined interfaces that are intuitive and easy to understand. Changes to these interfaces require modifications to all applications that depend on the particular interface, even if they do not depend on the operation or argument that was added or changed. Payload semantics, on the other hand, result in systems where changes to message formats can have a potentially lesser impact on the different components of the system. New functionality can easily be added to a system by creating new messages types. Consumers that are not dependent on the new messages types remain unaltered. Thus, payload semantics results in a weaker coupling at the type level.

The choice of *interface semantics* versus *payload semantics* is not an obvious one, as each approach has its pros and cons. Strongly typed languages, such as Java, limit the flexibility of the programmer by applying strict type checking at compile time. Almost all dependencies caused by the change of a type in the system can be discovered at compile time, thus significantly reducing the number of runtime errors. Weakly typed languages, such as TCL, offer much more flexible data manipulation, often based on string manipulation. These types of languages are generally used for scripting, especially in Web

Page 67

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 3.5. Tight Versus Loose Coupling

Recently, a lot of attention has focused on comparisons between *loose coupling* and *tight coupling* approaches to application interactions. On the technology side, this has mainly been driven by the potential of Web services to dynamically discover and bind to other services, such as through UDDI (Universal Description, Discovery and Integration). On the business side, this has been driven by the growing need of enterprises to increase flexibility with respect to changes in their own business processes and the ways in which they interact with partner companies.

Traditionally, business processes have been designed within the boundaries of an enterprise, or even within the different business units of the enterprise. These activities were managed with the help of detailed, real-time information. Processes that span multiple business units or enterprises typically have to deal with a very different set of requirements, needing a higher degree of flexibility. In these kinds of scenarios, one sees a much higher degree of uncertainty, a much more frequent change in terms of participants and their roles, and a constant evolution of the types of interactions required.

There appears to be a consensus that for these types of "in-flux" situations to operate, a loosely coupled architecture is required because loose coupling is seen as helping to reduce the overall complexity and dependencies. Using loose coupling makes the application landscape more agile, enables quicker change, and reduces risk. In addition, system maintenance becomes much easier. Loose coupling becomes particularly important in the B2B world, where business entities must be able to interact independently. The relationships between business partners often change rapidlyalliances are settled and cancelled, and business processes between trading partners are adopted to new market requirements. Two companies that are partners in one market might be competitors in another market. Therefore, it is essential that the underlying IT infrastructure reflect this need for flexibility and independence. Ideally, no business relationship should impact anothernew business relationships should be able to be established without any effect on existing ones. Functionality that is offered to one business partner might not necessarily be available to others. A change that is relevant for one business partner should have no impact on other partners. One trading partner may not cause another to block while waiting for a synchronous response, nor may one IT system depend on the technical availability of the IT system of a business partner.

The term *coupling* refers to the act of joining things together, such as the links of a chain. In the software world, *coupling* typically refers to the degree to which software components depend upon each other. However, the remaining question is: "*What are these dependencies, and to what degree can one apply the properties of* tight *and* loose?" Software coupling can happen on many different levels. One of the first issues is to differentiate between build time (compile time) dependencies and runtime dependencies. However, this is typically only sufficient when looking at monolithic applications. In a distributed environment, we believe that in order to determine the degree of coupling in a system, one needs to look at different levels. <u>Table 3-1</u> provides an overview of these levels and shows how they relate to the tight versus loose coupling debate.

Table 3-1	Tight	Versus	Loose	Coupling
-----------	-------	--------	-------	----------

Level	Tight Coupling	Loose Coupling	
Physical coupling	Direct physical link required	Physical intermediary	
Communication style	Synchronous	Asynchronous	
Type system	Strong type system (e.g., interface semantics)	Weak type system (e.g., payload semantics)	
Interaction pattern	OO-style navigation of complex object trees	Data-centric, self-contained messages	
Control of process logic	Central control of process logic	Distributed logic components	

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

## 3.6. Conclusion

Today's enterprise application landscapes are characterized by a variety of different technologies and concepts for distribution. On one hand, this variety arises within the enterprise organization itself for historical reasons, personal preferences of different people, and the dynamics of acquisitions and mergers. As a matter of fact, many redundant concepts exist within the same organizational unit. On the other hand, complementary concepts and technologies also exist. Due to the requirements of different types of distribution problems that coexist in one corporation, different solutions arise as well.

A modern architecture must be able to embrace all these technologies and concepts. Heterogeneityincluding heterogeneity of middlewaremust be understood as a fundamental fact that cannot be fought but instead must be managed. Furthermore, an architecture must accommodate frequent changes of the underlying distribution infrastructure. As a matter of fact, the lifecycles of today's infrastructure products are largely incompatible with the lifecycles of enterprise applications. Thus, you must protect the assets of an existing application landscape and simultaneously take advantage of the latest infrastructure products.

In this chapter, we have discussed the necessity of carefully choosing the right approach to integrating two distributed software components. Among other issues, you must decide on the appropriate communication infrastructure, synchrony, call semantics, usage of an intermediary, and object-oriented versus data-centric interfaces. All these decisions impact the coupling of the two systems.

#### References

[Bras2002] Braswell, Byron, George Forshay, and Juan Manuel Martinez . *IBM Web-to-Host Integration Solutions*, 4th ed. IBM Redbook SG24-5237-03, 2002.

[Lon1999] Long, Rick, Jouko Jäntti, Robert Hain, Niel Kenyon, Martin Owens, and André Schoeman . *IMS e-business Connect Using the IMS Connectors*. IBM Redbook SG24-5427-00, 1999.

[Tan2003] Tanenbaum, Andrew S. Computer Networks, 4th ed. Prentice-Hall, 2003.

[Tan2002] Tanenbaum, Andrew S. and Maarten van Steen . *Distributed Systems: Principles and Paradigms*. Prentice-Hall, 2002.

[Cou2001] Coulouris, George, J. Dollimore, and T. Kindberg . *Distributed Systems Concepts and Design*, 3rd ed. Addison-Wesley, 2001.

[Rei1992] Reisig, Wolfgang . A Primer in Petri Net Design. New York: Springer Compass International, 1992.

#### URLs

http://www.rfc-editor.org

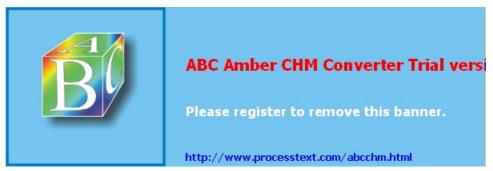
http://www.omg.org

http://www.microsoft.com/com

http://java.sun.com/j2ee

http://www.bea.com Team LiB

▲ PREVIOUS NEXT ▶



# **Part I: Architectural Roadmap**

The first part of this book describes the architectural roadmap to the service-enabled enterprise. We define the term Service-Oriented Architecture (SOA), provide a classification of service types, describe the different expansion stages of a SOA, show how to address key issues such as business processes and transactions, outline a service bus infrastructure, and describe how to use the different SOA concepts in real-world applications.

#### Team LiB

▲ PREVIOUS NEXT ▶



# **Chapter 4. Service-Oriented Architectures**

This chapter provides a definition of Service-Oriented Architecture and introduces its key conceptsnamely, application frontend, service, service repository, and service bus. It lays the conceptual foundation for a more in-depth discussion about the ways in which SOAs help address the specific issues of enterprise applications that are covered in <u>Chapter 5</u>, "Services as Building Blocks," and Chapter 6, "The Architectural Roadmap."

Section 4.1 gives a general definition of the term *architecture* as we use it throughout this book. Section 4.2 defines the term *Service-Oriented Architecture*. Section 4.3 describes elements of a Service-Oriented Architecture, such as application frontends, services, the service repository, and the service bus in detail.

#### Team LiB

♦ PREVIOUS NEXT ►



### 4.1. What Is a Software Architecture?

The literature in our field provides many different definitions of software architecture. Booch, Rumbaugh, and Jacobson [BRJ99] claim that "*An architecture is the set of significant decisions about the organization of a software system* . . ." Brass, Clements, and Kazman define software architecture in [BCK03]: "*The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among <i>them*." The IEEE Standard 610.12-1990 claims that "*Architecture is the organizational structure of a system*." Fowler characterizes architecture in [Fow02]: "'*Architecture' is a term that lots of people try to define, with little agreement. There are two common elements: One is the highest-level breakdown of a system into its parts; the other, decisions that are hard to change.*" You can find even more definitions at http://www.sei.cmu.edu/architecture/definitions.html.

For our purposes, we define software architecture in the sidebar, "<u>Definition of Software</u> <u>Architecture</u>."

### **Definition of Software Architecture**

A software architecture is a set of statements that describe software components and assigns the functionality of the system to these components. It describes the technical structure, constraints, and characteristics of the components and the interfaces between them. The architecture is the blueprint for the system and therefore the implicit high-level plan for its construction.

In this book, we will also use the terms *application* and *application landscape*. An application is a set of software components that serves a distinctive purpose, and an application landscape is the sum of all applications of an organization. Ideally, all applications of an application landscape comply with a single architectural blueprint. However, in practice, they usually don't. We also casually use one particular phrase: *"Software component X belongs to architecture Y."* More precisely, this phrase means: *"Software component X belongs to an application landscape which is designed according to an architecture Y."* 

#### Team LiB

♦ PREVIOUS NEXT ►



Team LiB

### 4.2. What Is a Service-Oriented Architecture?

Now we introduce the basic concepts of SOAs as we use them in the remainder of this book. As we previously emphasized, this book focuses on enterprise architectures and their specific characteristics. Consequently, we will also discuss the specific characteristics of SOAs.

As we mentioned earlier, an SOA is based on four key abstractions: *application frontend*, *service, service repository*, and *service bus* (see Figure 4-1). Although the application frontend is the owner of the business process, services provide business functionality that the application frontends and other services can use. A service consists of an implementation that provides business logic and data, a service contract that specifies the functionality, usage, and constraints for a client. of the service, and a service interface that physically exposes the functionality. The service repository stores the service contracts of the individual services of an SOA, and the service bus interconnects the application frontends and services.

n A client can either be an application frontend or another service.

# Figure 4-1. Services and application frontends are the major artifacts of an SOA. In addition, we also have a service repository and service bus.

### **Definition of Service-Oriented Architecture**

A Service-Oriented Architecture (SOA) is a software architecture that is based on the key concepts of an application frontend, service, service repository, and service bus. A service consists of a contract, one or more interfaces, and an implementation.

The whole concept of an SOA focuses on the definition of a business infrastructure. When we use the term "service," we have in mind a business service such as making airline reservations or getting access to a company's customer database. These services provide business operations such as *get reservation, cancel booking*, or *get customer profile*. Unlike business services, technical infrastructure services, such as a persistency service or a transaction service, provide operations such as *begin transaction, update data*, or *open cursor*. Although this kind of technical functionality is very useful when it comes to implementing a business operation, it has little strategic relevance from the SOA point of view. More generally, technology must not have any impact on the high-level structure of the application landscape or cause dependencies between components. Actually, the SOA must decouple business applications from technical services and make the enterprise independent of a specific technical implementation or infrastructure.

The application frontends are the active elements of the SOA, delivering the value of the SOA to the end users. Nevertheless, you must always take into account that the services provide structure to the SOA. Although the services can often remain unaltered, the application frontends are subject to change, as are the business processes of the enterprises. Consequently, the lifecycle of application frontends is much shorter than the lifecycle of the underlying services. This is why we regard services as the primary entities of strategic importance in an SOA (see Figure 4-2).

# Figure 4-2. The estimated lifecycles of data, services, application frontends, and technologies are different.

[View full size image]

### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

### **4.3. Elements of a Service-Oriented Architecture**

In this section, we take a closer look at the key elements of the SOA, including the application frontend, services, the service repository, and the service bus.

### **4.3.1. APPLICATION FRONTENDS**

Application frontends are the active players of an SOA. They initiate and control all activity of the enterprise systems. There are different types of application frontends. An application frontend with a graphical user interface, such as a Web application or a rich client that interacts directly with end users, is the most obvious example. However, application front-ends do not necessarily have to interact directly with end users. Batch programs or long-living processes that invoke functionality periodically or as a result of specific events are also valid examples of application frontends.

Nevertheless, it is entirely possible that an application frontend delegates much of its responsibility for a business process to one or more services. Ultimately, however, it is always an application frontend that initiates a business process and receives the results.

Application frontends are similar to the upper layers of traditional multilayer applications. Although you might expect that services more closely resemble the lower layers, this is not the case. The following chapters demonstrate that services have a different structure, which is characterized by vertical slicing.

### 4.3.2. SERVICES

A service is a software component of distinctive functional meaning that typically encapsulates a high-level business concept. It consists of several parts (see Figure 4-3).

**Contract.** The service contract provides an informal specification of the purpose, functionality, constraints, and usage of the service. The form of this specification can vary, depending on the type of service. One non-mandatory element of the service contract is a formal interface definition based on languages such as IDL or WSDL. Although it is not mandatory, a formal service interface definition adds a significant benefit: It provides further abstraction and independence of technology, including programming language, middleware, network protocol, and runtime environment. However, it is important to understand that the service contract provides more information than a formal specification. The contract can impose detailed semantics on the functionality and parameters that is not subject to IDL or WSDL specifications. In reality, many projects must cope with services that cannot provide formal service interface descriptions.<sup>20</sup> In these cases, the service can deliver access libraries or a detailed technical description at the network protocol level. However, it is important to understand that every service requires a service contract protice available.

Notice that the key task of a project aiming to introduce SOAs at the enterprise level is often not to implement new business functionality, but rather to identify suitable existing application modules and components and wrap them with service interfaces with the appropriate level of functionality and granularity, thus making them available as services in an easier-to-use and better documented manner.

**Interface.** The functionality of the service is exposed by the service interface to clients that are connected to the service using a network. Although the description of the interface is part of the service contract, the physical implementation of the interface consists of service stubs, which are incorporated into the clients<sup>10</sup> of a service and dispatcher.

Application frontends or other services.

**Implementation.** The service implementation physically provides the required business logic and appropriate data. It is the technical realization that fulfills the service contract. The service implementation consists of one or more artifacts such as programs, configuration data, and databases.

**Business logic.** The business logic that is encapsulated by a service is part of its implementation. It is made available through service interfaces. However, programming against interfaces is desirable, whether or not one applies a service-oriented approach.

**Data.** A service can also include data. In particular, it is the purpose of a data-centric

### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

### 4.4. Conclusion

In this chapter, we introduced the key concepts of Service-Oriented Architecture.

We began our discussion with a general definition of software architecture: ". . . set of statements that describe software components and assigns the functionality of the system to these components. It describes the technical structure, constraints, and characteristics of the components and the interfaces between them . . ." The definition of an SOA is based on this definition. It states that "A Service-Oriented Architecture (SOA) is a software architecture that is based on the key concepts application frontend, service, service repository, and service bus."

### References

[BCK03] Bass, Len, Paul Clements, and Rick Kazman . *Software Architecture in Practice*. Addision-Wesley, 2003.

[BRJ99] Booch, Grady, James Rumbaugh, and Ivar Jacobson . *Unified Modeling Language User Guide*. Addision-Wesley, 1999.

[Fow02] Fowler, Martin . *Patterns of Enterprise Application Architecture*. Addision-Wesley, 2002.

### URLs

http://www.sei.cmu.edu/architecture/definitions.html

♦ PREVIOUS NEXT ►



♦ PREVIOUS NEXT ►

# **Chapter 5. Services as Building Blocks**

The focus of a Service-Oriented Architecture is on the functional infrastructure and its business services, not the technical infrastructure and its technical services. A business-oriented service in an SOA is typically concerned with one major aspect of the business, be it a business entity, a business function, or a business process. This chapter provides an in-depth discussion of the different types of services.

In <u>Section 5.1</u>, we establish a classification that distinguishes between basic, process-centric, intermediary, and public enterprise services, and we discuss the characteristics of these different classes of services and what this means from a design, implementation, and project management point of view. These service types have significantly different characteristics with respect to reusability, maintainability, scalability, and performance; therefore, it's crucial to understand them in order to implement them efficiently.

Section 5.2 introduces SOA layers for design at the application landscape level. As we will demonstrate, SOA layers are largely independent of the system architecture's tiers, which is a major benefit of the SOA approach.

#### Team LiB



Team LiB

## 5.1. Service Types

An important feature of a software architecture is that it breaks down the overall structure of a software system into smaller components. These components are intended to be flexible building blocks.

### 5.1.1. MOTIVATION

Being able to classify service types is a precondition for the effective design of SOAs.

**Common language.** Being able to talk about the specific nature of different services at an abstract level will enable the different stakeholders in an SOA projectbusiness analysts, architects, designers, managers, and programmersto communicate their ideas and concerns more effectively.

**Vertical slicing.** Classifying services according to their specific nature is a prerequisite to breaking down a complex application landscape into manageable parts. In an SOA, this will naturally lead to a "vertical slicing," which is an important part of SOA-centric project management (see Chapter 13, "SOA Project Management").

**Effective estimating.** The classification of services is extremely helpful when it comes to making proper estimates on their implementation and maintenance cost. These costs depend on the complexity of the implementation, the level of design for reuse, and the frequency of change. These factors will vary by service type.

**Separation of code segments with different reuse characteristics.** It is good practice to separate code that is supposed to be reused from other code that is unique to a single project. This separation improves the reusability of the "purified" code because it eliminates any project-specific ballast that would complicate reuse. It also helps you avoid fruitless efforts to make project-specific code fit for a reuse that will never happen. Being able to classify your services according to our classification matrix will enable a cleaner separation between reusable and once-off code.

**Choosing the right implementation strategy.** Different types of services require different implementation strategies. Choosing an unnecessarily complex implementation strategy for a simple service will naturally lead to inefficiencies. For example, services that maintain conversational state can be very helpful in order to simplify clients. The client implementation can be "thin," and the service can provide all the necessary business logic to support even complex business processes. However, stateful services often have a negative impact on the scalability of a distributed system. It is therefore advisable to identify services that inevitably require conversational state and separate them from other services.

**Managing change.** Finally, it is important to separate business logic that is exposed to a high frequency of change from business logic that is more stable. Doing so can significantly reduce the costs and risks of maintenance. Once again, our classification matrix will be helpful in identifying services with different change characteristics. It should be noted that this is good practice in any development situation, not just for SOAs. However, SOAs are particularly well suited to enable this kind of code separation.

### **5.1.2. CLASSIFICATION**

We differentiate between four classes of services: basic services, intermediary services, process-centric services, and public enterprise services. Figure 5-1 introduces a basic notion we will use throughout this book.

# Figure 5-1. We distinguish basic, process-centric, intermediary, and public enterprise services.

In the remainder of this book, we will also use the terms *participant* or *SOA participant*. These terms comprise both application frontends and services. <u>Table 5-1</u> provides an overview of the different service types and their key characteristics.

### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

### **5.2. Layers on the Enterprise Level**

In this chapter, we have already discussed different service types. Now we take a first look at the overall structure of an application landscape and how the services relate to each other.

Traditionally, software layers provide important levels of abstraction. You can assume that layers are sequentially ordered, where layer N is above layer N+1. Code segments within one layer N can use other code segments with the same layer in addition to code segments of the layers N+1. In a distributed environment, the concept of tiers exists, where a tier is a set of contiguous layers that can be deployed separately. Although layers and tiers are extremely useful abstractions for the construction of single applications (or services), neither is suited as an abstraction at the enterprise level. Service-Oriented Architectures provide application frontends and services that are much more suited to this purpose.

SOA layers, which you must not confuse with these traditional software layers, and tiers provide a conceptual structure at the enterprise level that organizes the application frontends and services (see Figure 5-9). Each layer contains distinct types of services and application frontends:

**Enterprise layer.** The top layer of SOAs contains application frontends and public enterprise services, which are the end-points that provide access to the SOA. These endpoints facilitate both the communication between end users and the SOA (application frontends) and enable cross-enterprise (or cross-business unit) integration (public enterprise services).

**Process layer.** The process layer contains process-centric services the most advanced service type.

**Intermediary layer.** The third layer contains intermediary services. These services act as façades, technology gateways, and adapters. You can also use an intermediary service in order to add functionality to an existing service.

**Basic layer.** The bottom layer contains the basic services of the SOA. Basic services represent the foundation of the SOA by providing the business logic and data. The basic layer also contains proxies for other companies' public enterprise services.

# Figure 5-9. No 1:1 relationship exists between traditional tiers and SOA layers. These concepts actually are largely independent.

As we will see in <u>Chapter 6</u>, "The Architectural Roadmap," many problems can be solved with two or three SOA layers. In these cases, there is no benefit to artificially introducing additional layers simply to have a "complete" SOA. Recall that an SOA is about simplification. As long as a problem can be solved with simple measures, it is best to do so.

Although we will not discuss this matter in great detail, we must briefly consider the deployment of SOAs and the resulting system architecture in order to prevent a common misunderstanding: SOA layers do not correspond to physical tiers. It is not necessary for services, which originate from different SOA layers, to be deployed at different tiers. Nor must all services of one SOA layer be deployed at the same location. The system architecture is driven by matters such as available hardware and system software, system management requirements, and compatibility. These issues are largely independent of requirements such as maintainability or simplicity that drive the design of the services.

Actually, the design of the SOA and the system architecture are largely independent aspects of the application landscape, which is the remarkable strength of the SOA paradigm.

### **Decouple System and Software Architecture**

A major benefit of the SOA paradigm is to be able to design the system and the software architecture largely independently of each other. This results in a high

### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

### 5.3. Conclusion

In this chapter, we have introduced four classes of services: basic, process-centric, intermediary, and public enterprise services. These different services are predominantly used by the application frontends, which are the active elements of the SOA but are not services themselves. It is important for a designer of a Service-Oriented Architecture to be aware of the distinctive characteristics of the different service types because they all have different requirements with respect to design, implementation, operation, and maintenance.

In addition, this service classification is ideally suited to provide an overall structure within the SOA that is expressed by the different SOA layers, including enterprise layer, process layer, intermediary layer, and basic layer.

### References

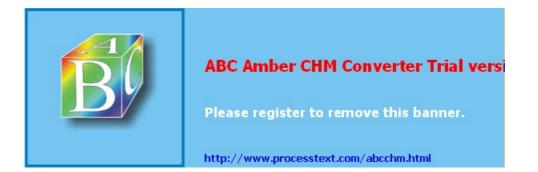
[GHJV95] Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides . *Design Patterns*. Addison-Wesley, 1995.

[Her2000] Herzum, Peter and Oliver Sims . *Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*. OMG Press, 2000.

[Rei1992] Reisig, Wolfgang . A Primer in Petri Net Design. New York: Springer Compass International 1992.

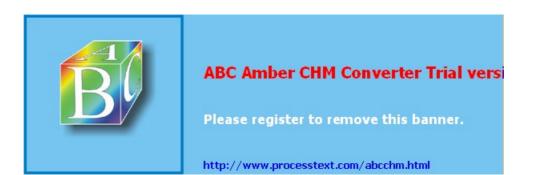
Team LiB

♦ PREVIOUS NEXT ►



# **Chapter 6. The Architectural Roadmap**

Implementing an SOA at the enterprise level is a significant endeavor that is likely to take many years. During this time, we probably will encounter many obstacles and will have to cope with frequently changing requirements. It is therefore essential that we look at a realistic roadmap for rolling out our Service-Oriented Architecture. It is important that our architecture *and* our roadmap is designed to cope with the complexity and dynamics of an enterprise environment. <u>Chapter 1</u> introduced an SOA-based *enterprise IT renovation roadmap*. In this chapter, we will focus on the architectural aspects of this roadmap. In Chapter 12 of this book, we will then focus on the political and organizational aspects.



Team LiB

### 6.1. The Architectural Roadmap

For the architectural roadmap, we have identified three expansion stages that signify different levels of maturity of an SOA. The expansion stages indicate the allocation of responsibility between the application frontends and the services (see Figure 6-1). The first stage is called *fundamental SOA*. In this expansion stage, much of the complexity and responsibility is still allocated at the application frontend. Although a fundamental SOA does not provide all features of a fully leveraged SOA, it is already a useful platform for an enterprise application landscape. The next expansion stage is called *networked SOA*. At this stage, intermediary services aggregate low-level basic services to more sophisticated services. Finally, we have *process-enabled SOAs*. At this stage, the application frontends delegate process control to the SOA.

### Figure 6-1. The expansion stages fundamental SOA, networked SOA, and process-enabled SOA are milestones of the architectural roadmap to the service-enabled enterprise.

In Figure 6-2, we depict the typical development of an application. It shows how permanent change requests and a lack of continuously applied architectural concept diminish maintainability and how an SOA-driven refactoring can reestablish agility. In Part III of this book, we present several case studies that show how different enterprises experienced difficulties when extending their historically grown application landscapes and how a transition to a Service-Oriented Architecture helped to incrementally overcome their difficulties.

### Figure 6-2. Agony versus agility.

[View full size image]

It should be noted that the concept of expansion stages cannot be applied in a black-and-white fashion. As soon as the SOA is established, you will probably find areas of differing maturity that are at a different expansion stage within itfortunately, without any undesired impact on the SOA itself. This reveals a major benefit of Service-Oriented Architecturesthey enable enterprises to start small and evolve in small steps as required in future projects. Actually, the SOA enables both evolutionary development of technology and functionality. Furthermore, the SOA supports different developments that run in parallel or in sequence. The SOA brings all these developments together in a smooth fashion, without harming a single project or the overall SOA endeavor. However, the expansion stages differ in regard of the distribution of responsibilities between application frontends and services. With an increasing maturation of the SOA, the services gain more and more responsibilities.

### Allow Different Expansion Stages

Typically, different expansion stages coexist in the same SOA. Do not fight this heterogeneity! Develop different areas of the SOA in parallel to the requirements of business-driven projects.

The definition of your SOA strategy and the required level of maturity strongly depend on the scope of the business integration you are planning to reach. Obviously, implementing an SOA strategy is always a question of budget, and the further you are planning to advance your SOA, the longer you will have to invest (see also <u>Chapter 12</u>, "The Organizational SOA Roadmap," for a discussion on the budget requirements of an SOA).

Identifying the required scope of the business integration is usually a good first step on the way toward the definition of the overall SOA strategy because there is a strong correlation between the integration level one is aiming for on the one hand and the required maturity level of the SOA on the other. Figure 6-3 depicts this dependency.

Figure C.D. The methods is a fall a COA with mean at the the sum and is a

Page 94

### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

### 6.2. Fundamental SOA

A *fundamental SOA* consists of two layers: the basic layer and the enterprise layer. Distinguishing these two layers helps single applications to define a proper high-level structure. It also enables two or more applications to share business-logic and live data. Although a fundamental SOA represents a rather simple approach, it provides a strong platform for large enterprise application landscapes. In fact, it is a major improvement compared to many of today's real-world scenarios. It is also an excellent starting point for the introduction of an enterprise-wide SOA because it enables enterprise organizations to start small on the technical side and focus on other critical success factors (see <u>Chapter 12</u>, "The Organizational SOA Roadmap").

A simple example (see Figure 6-4) shows how one application can be divided into meaningful components using an SOA. The Airline Web site utilizes four services that encapsulate the major business entities and their behaviors that are relevant to the business processes that are exposed to the customers.

# Figure 6-4. A fundamental SOA consists of two layers. Application frontends and basic services provide a fully functional base for enterprise applications.

[View full size image]

We can now expand the original example by adding another application. A key characteristic of a fundamental SOA is that it enables two or more applications to share live data and business logic. In our example, we consider a traditional billing application that is required to keep track of terms of payment and the handling of demand notes (see Figure 6-5). This type of application is traditionally provided with customer and billing data through nightly batch jobs. Integrating billing and invoicing into a fundamental SOA enables the booking and the billing applications to share live data. In practice, u this means that the billing application gets access to the customer service and billing services that, in turn, make batch processes that transfer data between the applications obsolete. 2 In addition, there are clear benefits for the booking system. As a result of up-to-date billing, the booking system also has access to precise, up-to-date data at any time. This increases the capability of the booking system to treat customer requests in a more appropriate manner. For example, a change in the credibility status of a customer, which is detected by the billing system, is instantly available for the booking system. As previously mentioned, in many traditional environments, this information is only available to the databases of the booking application after nightly batch updates.

It should be noted that a real-world scenario would be much more complex, as depicted in Figure 6-5. It would involve a customer care application, a sort of workbasket, and additional business services. However, the aforementioned benefits still apply.

<sup>22</sup> Note that an update in one application is instantly visible in the other application. In a traditional EAI scenario, you should consider decoupling the original data change from notifying the second application due to throughput and performance considerations. In the SOA world, these notifications are obsolete.

# Figure 6-5. Enterprise application integration becomes obsolete if several applications share live data.

[View full size image]

The introduction of a fundamental SOA is the first important step toward a truly SOA-enabled enterprise. The following summarizes the main characteristics and scope of a fundamental SOA:

- A fundamental SOA is an appropriate base for an enterprise application landscape.
- Due to its simplicity, it is technically easy to implement.
- It is a good starting point for an SOA that enables the introduction of more advanced expansion stages in the future.
- The application frontends are still complex. They must cope with the control of

### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

### 6.3. Networked SOA

The next expansion stage is called *networked SOA*, and it deals with backend complexity in addition to technical and conceptual integration. It includes a layer of intermediary services that can include *façades, technology gateways, adapters,* and *functionality adding services*.

We start our discussion with *façades*. As we discussed in <u>Chapter 5</u>, "Services as Building Blocks," façades can serve various needs. Most importantly, they encapsulate some of the complexity of underlying basic services by providing an aggregated API that enables clients to more easily utilize the functionality of the basic layer. In <u>Chapter 8</u>, "<u>Process</u> <u>Integrity</u>," we will discuss one aspect of this complexitythe challenges of achieving process consistency. Introducing a façade is one possible solution to this issue. Figure 6-6 provides an example of the booking and the billing services that must update their databases in a coordinated manner. Depending on the detailed requirements and the concrete technology upon which the booking and the billing services are built, it is not always simple to guarantee consistency. Actually, you will want to shield client developers from this kind of complexity. It is often necessary to apply particular skills in server-side development to master the challenges of the design and implementation tasks. Moreover, if multiple clients require similar functionality, there should be no duplication of workload. Thus, in our example, this complexity is encapsulated by the intermediary service "BookAndBill."

# Figure 6-6. The intermediary service BookAndBill encapsulates the handling of distributed transactions that span the services Booking and Billing.

[View full size image]

Utilizing *technology gateways* is a handy technique to enable one service to be used in different technical environments.<sup>III</sup> In Figure 6-7, we describe a scenario in which the flight service that is implemented on CICS is exposed to EJB, .NET, and MQSeries environments by technology gateways. This enables developers to specify, design, develop, test, operate, and maintain exactly one instance of the flight service, including live data, and reuse it in many heterogeneous environments at the same time.

<sup>ISI</sup> Note that from a purely technical point of view, there is no difference between a technology gateway and an additional interface to the underlying basic service. Thus, it is possible to add some Java classes to an existing PL/I based service in order to provide an additional interfacing technology. The communication between the Java code and the PL/I code would become an internal matter of the service. The main difference between these two approaches is at the project management and team organization level. Most often, it is advisable to separate these technically different artifacts. In practice, you will find different subteams working on the different areas anyway. Separating their work by means of a service contract, configuration management, and distinct entries in the service repository is generally preferable. The same discussion applies to the adapters discussed in the following paragraphs.

# Figure 6-7. Technology gateways expose the functionality of services in technologically different environments.

[View full size image]

The perfect world does not need technology gateways. In the first place, you would not encounter heterogeneous technologies. Secondly, given that the clients are heterogeneous in the real world, you would choose a single uniform bridging technology to integrate all clients. In our example, you might want to implement a type of XML RPC interface directly as part of the CICS-based service and use this interface in all client environments. Unfortunately, the ongoing evolution of technology can raise many unforeseen issues and new requirements. You might want to adopt current improvements without reimplementing existing services. Furthermore, you cannot predict which technologies you will need to integrate in the future. It is also unclear which political or commercial constraints will have an impact on a technology decision at a specific point in time. As a matter of fact, the SOA paradigm enables the creation of clear designs that cope with this kind of heterogeneity. Although it is not desirable to have multiple technology gateways that provide access to the same service, these technology gateways do no real harm to the architecture.

Figure 6-8 depicts a typical chronology. It shows the major milestones of a check-in application after its launch in 1986. The first milestone was the integration with partner airlines. Besides other requirements, the partner airlines needed access to flight data. This

### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

### 6.4. Process-Enabled SOA

The third expansion stage is the fully leveraged SOA. The key feature of the *process-enabled SOA* is the maintenance of process state in process-centric services.

Similar to intermediary services, a process-centric service is both client and server in an SOA. The major difference between these service types is the fact that process-centric services are stateful. This is a crucial difference because handling state is a critical issue for server-side software. There are several possible reasons for introducing a process-centric service:

- Encapsulating the complexity of processes
- Sharing state between multiple clients
- Handling long-living processes

Encapsulating process state in a process-centric service enables application frontends to be simple and lightweight and at the same time very user-friendly with an elaborate handling of the user session.

Figure 6-11 extends the booking example first introduced in Figure 6-5. A new service "Booking process" encapsulates the business process "Booking." The Booking process utilizes the BookAndBill service and the Customer service. Although most of the work is carried out by the Booking service, the application frontend has access to all layers of the SOA.

### Figure 6-11. A fully developed process-enabled SOA has four layers.

[View full size image]

The scenario in Figure 6-11 is the straightforward evolutionary step arising from the scenario in Figure 6-5. However, a green-field design would be different. In our example, the BookAndBill façade could become part of the process-centric service. Omitting the BookAndBill service (see Figure 6-12) reduces our scenario to three layers. Because one of the primary goals of SOAs is simplicity, this is a desirable step. Furthermore, reducing the number of tiers between the application frontend and basic layer reduces the system's latency and the number of elements that can potentially fail.

# Figure 6-12. The Booking process encapsulates all functionality necessary to book flights. It also maintains the session state.

[View full size image]

But there are also possible scenarios where our BookAndBill service can be beneficial. Assume that several clients, such as a Web site, a B2B gateway, and a mobile application, are running simultaneously (see Figure 6-13). Typically, these applications require a distinct process-centric service. Factoring out the BookAndBill functionality becomes a reasonable design decision in this case.

# Figure 6-13. Several processes can utilize the BookAndBill service at the same time.

[View full size image]

However, as depicted in <u>Figure 6-14</u>, an alternative design exists that factors out common booking process functionality to a process-centric service that can incorporate the BookAndBill functionality. The shared process-centric service both maintains a channel-independent process state and shields the channel-specific process-centric services from backend complexity.

## Figure 6-14. In general, every channel of a multichannel architecture

Page 10

### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

### 6.5. Conclusion

In the previous chapter, we introduced four SOA layers that serve as a conceptual construct that enables the efficient organization of enterprise application landscapes.

Based on the concept of SOA layers, we distinguished three expansion stages that define the level of sophistication an SOA has achieved. The first expansion stagefundamental SOAdelivers a maintainable business infrastructure that provides the application landscape with flexible building blocks containing the business logic and data of the enterprise. While the integration of these building blocks is still allocated at the application frontend in the first expansion stage, the second stagenetworked SOAencapsulates this kind of complexity in an intermediary layer. Finally, the process-enabled SOA depicts the truly SOA-enabled enterprise, leveraging support for the business processes of the enterprise in process-centric services.

#### Team LiB

◀ PREVIOUS NEXT ►



# Chapter 7. SOA and Business Process Management

In previous chapters, we discussed the general "renovation roadmap" that describes the evolution of an enterprise IT infrastructure toward a more agile Service-Oriented Architecture. We examined fundamental and networked SOAs as the initial stages in great detail. According to our roadmap, the final stage in this evolution are process-enabled SOAs. Although process-centric services can be realized in many different ways (and can thus take many different shapes and forms), *business process management* (BPM) represents possibly the most consequent approach to process-enabling an SOA. Consequently, we provide a general introduction to BPM in this chapter, followed by a discussion on how BPM fits into the SOA landscape. In this chapter, we focus more on the technical (computational and architectural) aspects, while <u>Chapters 12</u>, "The Organizational SOA Roadmap," and <u>Chapter 13</u>, "SOA-Driven Project Management," concentrate on the IT strategy and project-management aspects.

#### Team LiB

♦ PREVIOUS NEXT ►



Team LiB

## 7.1. Introduction to BPM

Business process management has a number of predecessors. Following the *Total Quality Management* wave of the late 1980s, a new paradigm emerged in the early 1990s: *Business Process Reengineering* (BPR). In 1993, Michael Hammer and James Champy published their New York Times bestseller *Reengineering the Corporation* [HC93], which was followed by a plethora of other management books on the topic of process-orientation and reengineering. The promise of reengineering was to deliver dramatic business performance improvements in a relatively short period of time by completely reinventing existing business processes, that is, starting from scratch with new, optimized ways of doing business, throwing out the old, encrusted, inefficient procedures of the past.

However, after the initial BPR boom (and millions of dollars spent on management consultancies to lead BPR projects in Fortune 500 companies), the process movement became idle. Many projects resulted in complete failure, with experts claiming that between 60% and 70% of reengineering efforts failing to achieve expected results. Hammer and others attribute these failure rates to resistance to change, lack of understanding of the business models and underlying processes, and failure of nerve on the part of the client companies.

Almost ten years after BPR, you can observe a revival of the process movement under the umbrella term *business process management* (BPM), as advocated by Howard Smith and Peter Fingar in *BPM: The Third Wave* [SF03]. When comparing BPR and BPM, it appears as if one thing has fundamentally changed: While in the 1990s *reengineering* meant "starting from scratch," *process management* builds on and transforms that which already existsit recommends incremental change and evolutionary optimization. However, BPM is still about processes, which are the main competitive differentiator in all business activity.

BPM is a general management topic that focuses on the strategic and operational aspects of *process orientation* in a given business area. Mapping a BPM model onto an enterprise IT landscape is a challenging task, which has some interesting technical as well as IT management-related aspects.

### 7.1.1. BPM VERSUS BPMS

When discussing BPM, it is important to differentiate between the business and IT sides. When looking at the business side of BPM, you will often find related keywords such as ISO 9000 and Six Sigma. The IT side of BPM is often accompanied by keywords such as process modeling and workflow management (see Figure 7-1).

# Figure 7-1. IT and business people have different views of the processes of an organization.

[View full size image]

A BPMS (Business Process Management System) provides the technical platform for realizing BPM management initiatives. It comprises several parts including a BPM engine, facilities for business process monitoring, design tools and facilities for simulation. A BPMS installation can include several products or custom made software components. Closing the gap between the business and IT sides of BPM (or other process-oriented approaches) has been something of a holy grail in IT for two decades. Currently, it seems that the solution might be a conversion between more software engineering approaches such as CASE (Computer Aided Software Design) and MDA (Model Driven Architectures) on one hand, and workflow management and BPM approaches on the other (see [Fra03]).

BPM introduces the concept of "process processing" and stresses that this concept is not limited to the automatic execution of digital process models, but *"encompasses the discovery, design, and deployment of business processes, as well as the executive, administrative, and supervisory control over them to ensure that they remain compliant with business objectives"* [SF03]. This describes at a high level the features that are typically included in BPMS, a new software category that supports the entire lifecycle of modeling, executing, and monitoring business processes.

### 7.1.2. WHEN TO CHOOSE A BPMS

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 7.2. BPM and the Process-Enabled SOA

Having introduced the foundations of BPM and its long-term vision, we now take a closer look at what this means for the SOA architect.

## 7.2.1. THE PAST: DATA AND FUNCTIONS VERSUS OBJECTS VERSUS SERVICES

This chapter focused on BPM and process-orientation, which comes at the very end of our "enterprise renovation roadmap." We should now step back and look at the origins of SOAs and what can be learned from their evolution.

In the early days of functional programming, data and functionality were strictly separated. With the emergence of object orientation, people began to merge data and functionality into encapsulated, reusable object implementations. This worked particularly well for large, monolithic applications, such as complex graphical user interfaces. In the middle of the 1990s, people started to apply the concepts of object orientation to distributed systems. CORBA and a number of other standards for distributed object computing emerged. However, when applying distributed object technology in large-scale projects, it eventually became clear that this approach had some severe limitations. As a result, Service-Oriented Architectures emerged, with supporting technology platforms such as XML Web services.

So what problems with distributed objects led to the emergence of SOAs? The first problem that we usually cite is the fine level of granularity of distributed objects, which often leads to performance problems due to latency and other issues. An SOA addresses these performance issues by adopting patterns of more coarse-grained objects, which require less frequent interaction between clients and servers, thus reducing the number of network roundtrips, marshalling overhead, and so forth.

However, a second and potentially more critical problem exists: Due to the complicated interaction patterns that result from the fine-grained nature of distributed objects, the reuse of distributed objects became increasingly complex. The complex dependencies between different objects prevented efficient reuse and made these systems very inflexible and hard to maintain because few people really understood these dependencies and the impact that changes to individual objects and their interfaces would have on overall systems.

With SOAs, we take a deliberate step back from the highly complex, fine-grained, highly dependent distributed object models toward less complex, relatively coarse-grained, loosely coupled (i.e., less dependent) component interfaces.

## 7.2.2. THE FUTURE: CORE BUSINESS LOGIC VERSUS PROCESS CONTROL LOGIC

Rather than revert to a paradigm that separates data and functionality, the SOA should develop an architecture that differentiates between core business logic and process control logic. Both of these concepts comprise data and functionality, although in very different ways. Unfortunately, these concepts are not as clean as the concepts of data and functionality, but we believe that they are still essential for the successful implementation of an SOA-based "enterprise IT renovation roadmap." Let's take a look at each concept using concrete examples.

*Core business logic* comprises *basic data access services, complex calculations*, and *complex business rules*. Data access services represent core business logic because they make core business entities available to different systems. An example is a service that enables different applications to read and update shared customer data. An example of a complex calculation would be the calculation of an insurance premium based on statistical data that is encapsulated by the service. Different processes such as sales, premium adjustment, and risk assessment require this core business logic. It is not always clear whether business rules represent core business logic (residing in a basic service) or whether they should be a direct part of the process logic (e.g., residing in a BPMS or process-centric service). Often, this will depend on the level of complexity of the business rule. The simple rule that "all orders over USD 100,000 must be manually validated" might well be part of the process control logic. However, a complex claims validation engine that incorporates legislative data to check the validity of insurance claims would clearly fall into

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

## 7.3. Conclusion

As we saw in this chapter, an SOA represents a good foundation for adopting a process-oriented approach in the long term. You can introduce process orientation at different levels in an SOA, with the most powerful level being represented by a Business Process Management System (BPMS). However, migrating to a process-enabled SOA is a long process that can easily run for a number of years. In the meantime, enterprises must find ways to deal with processes and, in particular, process consistency in the short term. This issue is the focus for the next chapter.

#### References

[Fra03] Frankel, David S. *BPM and MDA: The Rise of Model-Driven Enterprise Systems*. Business Process Trends, http://www.businessprocesstrends.com/, June 2003.

[HC93] Hammer, Michael and James Champy . *Re-engineering the Corporation*. London: Nicholas Brealey, 1993.

[Mil80] Milner, Robin . "A Calculus of Communication Systems." *Lecture Notes in Computer Science*, volume 92, 1980.

[Rei1992] Reisig, Wolfgang . A Primer in Petri Net Design. New York: Springer Compass International, 1992.

[SF03] Smith, Howard and Peter Fingar . *BPM: The Third Wave*. Tampa: Meghan-Kiffer Pr., 2003.

#### URLs

http://www.bpmi.org

http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

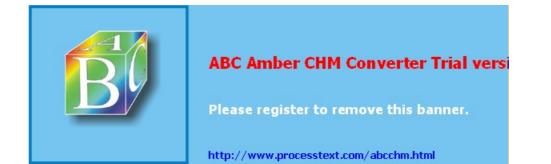
http://www-106.ibm.com/developerworks/library/ws-bpel/

http://www.gotdotnet.com/team/xml\_wsspecs/xlang-c/default.htm

http://www.businessprocesstrends.com/

Team LiB

◀ PREVIOUS NEXT ►



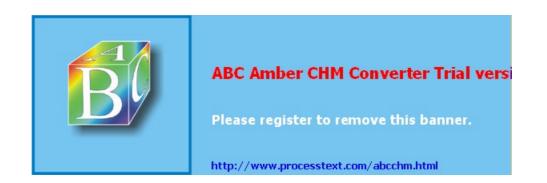
#### Team LiB

## **Chapter 8. Managing Process Integrity**

Achieving consistency in the execution of complex business processes that span multiple subsystems is one of the most challenging problems in IT. The first part of this chapter provides an overview of the problem scope, common solutions for achieving process integrity, and how they fit into an SOA. The second part of the chapter provides a set of concrete recommendations for SOA architects, based on an extension to our airline example.

#### Team LiB

♦ PREVIOUS NEXT ►



Team LiB

## 8.1. Data Versus Process Integrity

*Process integrity* is not necessarily a well-established or well-defined concept. The core building blocks of *process integrity* are based on the widely established concepts of *data integrity*. However, as we will see in the following, *data integrity* is insufficient for addressing all integrity requirements of complex business processes spanning multiple IT systems, which is why it is necessary to introduce *process integrity* as a concept that helps to address these more complex and demanding requirements.

#### 8.1.1. DATA INTEGRITY

Data integrity is an umbrella term that refers to the *consistency, accuracy*, and *correctness* of data. The classical mechanisms for ensuring data integrity are often closely tied to the concepts of relational databases. The primary types of data integrity include *entity, domain,* and *referential integrity*. Entity integrity requires that each row in the table be uniquely identified. Domain integrity requires that a set of data values fall within a specific range (domain)for example, a birth date should not be in the future. Referential integrity refers to the validity of relationships between different data tuples. Finally, *user-defined* data integrity refers to types of data integrity is typically enforced using a data access layer, triggers, and stored procedures. These are all fairly technical concepts, and typical business requirements for data integrity go far beyond technical concepts. These concepts are typically limited to a single database. As you will see in the following, more flexible concepts for ensuring process integrity are required if you are leaving the domain of a single database or application system.

#### 8.1.2. PROCESS INTEGRITY

The problem with complex business processes that span multiple IT systems goes beyond the issues of traditional data consistency. In these kinds of situations, we are not dealing with short-lived updates of data contained in a central repository, but instead with long-lived processes that cross multiple systems. These processes do not often have a well-defined state because it is not possible to obtain access to all the participants all the time, a requirement necessary to determine the process state. This is particularly true for processes that span the boundaries of business units or enterprises. Take the example of a manufacturer who receives product parts from a supplier. If the manufacturer receives a last-minute cancellation for an order, there will be time intervals where the internal systems reflect this cancellation while the order for related parts is still with the supplier. This is what we refer to as a *process inconsistency*.

#### 8.1.3. TECHNICAL FAILURES VERSUS BUSINESS EXCEPTIONS

The key to maintaining process integrity is to ensure that failures within or between the execution of the different steps that comprise a complex business process are captured and the necessary steps taken to resolve the problem. It is necessary to differentiate between technical failures on one hand and exceptions and special business cases on the other:

**Technical failures.** Technical failures include database crashes, network problems, and program logic violations, among many others. Often, these problems can be addressed in their respective context, for example through backup and recovery mechanisms (provided by the DBMS; e.g., transactions), retries (in the case of network problems), exception handlers (e.g., a Java catch clause), or process restarts (e.g., after a process terminated because of a C++ NULL pointer exception). However, in many cases, technical failures must be addressed using more complex, often custom-built solutions. For example, systems must cope with network problems by temporarily storing a process state locally until the subsystem to which it attempted to connect can be reached again.

**Business exceptions.** Business exceptions can range from very simple exceptions to arbitrarily complex ones. An example of a simple exception is an attempt by a customer to book a flight on a date that lies in the past. Such a simple *domain inconsistency* (see the previous discussion on data inconsistencies) can be addressed at the database level. For the sake of usability, it can be handled directly in the user interface (e.g., Java script in the browser). However, simple domain inconsistencies have local impact only. An example of a more complex business exceptionwith a more proliferating impactis an *out of stock* 

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

### 8.2. Technical Concepts and Solutions

You can choose from a wide range of solutions to implement process integrity. These solutions range from simple technical solutions such as distributed logging and tracing to advanced transaction concepts. BPM systems provide the facility to address process integrity on a less technical and more business-oriented level. BPMs are used to model and execute business processes, as we discussed in <u>Chapter 7</u>, "SOA and Business Process Management." They enable us not only to explicitly model special cases but also to provide definitions for the appropriate countermeasures in case of exceptions. We look at each of these solutions in turn, followed by recommendations for their application in an enterprise SOA.

#### 8.2.1. LOGGING AND T RACING

Logging and tracingat different levels of sophisticationis probably still the most commonly used approach for providing at least rudimentary levels of process integrity on an ad-hoc basis.

Log traces are commonly used for debugging but are also used in production systems in order to identify and solve problems in day-to-day operations. Particularly in the case of complex systems that integrate large numbers of non-transactional legacy systems, logging is often the only viable approach to providing a minimum level of process integrity, especially if processes or workflows are implemented implicitly (i.e., there is no dedicated BPM system). Often, the operators of these types of systems employ administrators that manually fix problems based on the analysis of log files. If the log file provides a complete trace of the steps executed in a particular process until the failure occurred, the administrator has some chance of fixing the problem, even if this often requires going directly to the database level to undo previous updates or fix some problem to enable the process to continue.

A key problem with logging-based problem analysis and repair is the lack of correlation between the different log entries that relate to a particular logical process instance, especially for processes that are implemented implicitly. If a process fails due to a technical fault, someone must identify what has happened so far in order to complete or undo the process. To do this, you must find the log entries that relate to the process, possibly across different log files from different systems. Ideally, some kind of correlation ID (related to process instances) should exist for each log entry because this helps with the log consolidation (as depicted in Figure 8-1). However, often the only way to correlate events is by comparing timestamps, which is a difficult task, especially with systems that use distributed log files. For example, how is it possible to relate a JDBC exception written into a local log by an EJB application server to a database deadlock event that was written to a database log (both log entries are potentially relevant for identifying and fixing the problem in the application)? In this case, a significant chance exists that both exceptions occurred at the same time. The problem becomes even more difficult if processes are long-lived and you are trying to find out which customer account was modified earlier by a process that failed later, such as when updating a shipping order.

#### Figure 8-1. Consolidated logs can help system administrators deal with error situations in distributed, long-lived processes. Log consolidation can be performed across systems (e.g., providing the ability to relate updates in one database to updates in another), as well as within systems (e.g., relating database logs to application server logs).

Nevertheless, logging and tracing is still important in many operational systems and is often the only possible way to achieve at least minimum process integrity. Many projects have therefore invested heavily in building a sophisticated infrastructure that helps with distributed logging and log analysis. Often, this infrastructure is built on or tightly integrated with system management platforms such as IBM Tivoli or HP Openview.

Chapter 9, "Infrastructure of a Service Bus," provides a detailed overview of how this can

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 8.3. Recommendations for SOA Architects

Having discussed the concepts of data and process integrity on a more conceptual level, it is now time to examine concrete recommendations for SOA architects. In order to make this discussion as hands-on as possible, we will first introduce an extension to our airline example, which will serve as a basis for the discussion that follows.

#### 8.3.1. EXAMPLE SCENARIO: TRAVEL ITINERARY MANAGEMENT

The management team of our airline has decided to expand the current online offering by providing airline customers with the ability to not only book individual flights but also to create complete itineraries for their trips, including multiple flight, hotel, and car reservations.

The new itinerary management system will reuse and build upon several of the airline's existing IT systems, in particular the customer management and billing systems. In addition, the customer database will be expanded to support the management of complex itineraries. Management decided that two different frontends are required for the new system: a Web-based online frontend and a call center for telephone support of customers. A further decision was to use different technologies in each of the two frontends: the Web-based frontend will use thin HTML clients, whereas the frontend for the call center agents needs more complex functionality and will thus be implemented as a VB GUI (fat client).

The high-level architecture of the system is based on three backend services: customer management (including itineraries), billing, and complex processes (in order to manage customer complaints regarding itineraries or invoices). In addition, a number of partner systems will have to be integrated, including partner airlines' flight reservation systems and hotel and car reservation systems (although this is not a key aspect of this discussion). Figure 8-5 provides an overview of the system architecture.

## Figure 8-5. The new travel itinerary management system provides services to manage customers, itineraries, invoicing, and complex incidents.

[View full size image]

We will look at two key transactions throughout the rest of this chapter: *confirm itinerary* and *create invoice. Confirm itinerary* is an important transaction of the customer management system, which is responsible for confirming the individual flight, hotel, and car reservations on an itinerary, involving potentially complex interactions with partner systems. This transaction is potentially irreversible, in that the system might require a cancellation fee when attempting to cancel a previously confirmed booking (assuming that a customer has proven creditworthiness, the system creates an invoice, calculates the total amount and taxes for each individual item on the itinerary, and sends a letter with a printed version of the invoice to the customer by mail.

These two transactions are interesting for several reasons. First, they are closely related at the business level because the confirmation of an invoice inevitably causes costs and therefore must be accompanied by a valid invoice under all circumstances. Second, these transactions cross several organizational boundaries because they use two services provided by different departments (customer management is a marketing function, and invoicing is a back-office function) and even involve several different companies (other airlines, hotels, and car rental companies). Finally, these transactions also cross several technical boundaries. Notice in particular that in our airline example, these two transactions are related to two independent databases.

## 8.3.2. OPTIMISTIC CONCURRENCY CONTROL SHOULD BE THE DEFAULT

To begin with, it is necessary to explain how to deal with concurrent access to shared data in an SOA. Two widely established models for managing concurrent access to shared data by multiple users exist: *optimistic* and *pessimistic concurrency control*. Both come in

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

### 8.4. Conclusion

Ensuring process integrity is often more a project management issue than a technical issuethe available technical solutions are relatively well known, but how and when to select each solution is often a key problem. Finding the right tradeoff between integrity requirements from a business perspective and sophistication of the technical solution is a task that requires careful analysis from responsible architects and project managers.

It is important for project managers to understand that a very clear tradeoff exists between the level of process integrity and implementation costs. For example, while the two-phase commit protocol potentially provides a very high level of integrity (at least in homogeneous, tightly coupled environments), it comes with very high implementation costs because it requires highly skilled architects and developers and very expensive software. However, transactional steps incur lower costs and offer reasonable process integrity properties. They often provide a very good cost/integrity tradeoff (<u>Figure 8-18</u> categorizes several approaches to process integrity in regard to their cost integrity ratio).

#### Figure 8-18. It is key for project managers to realize the tradeoff between the level of process integrity provided by a particular technical solution and its implementation costs.

[View full size image]

If technical project managers can communicate these tradeoffs clearly to the decision makers at the business level, they enable them to make judgments regarding the level of consistency requirements they have and the money they are prepared to spend on them. <u>Chapter 13</u>, "<u>SOA Project Management</u>," adds to this discussion by providing concrete tools for project managers to enable them to find the right tradeoff between implementation costs and integrity requirements.

#### References

[Co70]Codd, E. F. *A Relational Model of Data for Large Shared Data Banks*. ACM Communications, Volume 13, No 6, June 1970.

#### Team LiB

♦ PREVIOUS NEXT ▶



4 PREVIOUS NEXT +

## Chapter 9. Infrastructure of the Service Bus

In this chapter, we describe the different elements that constitute an SOA-enabling infrastructure, referred to in the following as a *service bus*. One of the fundamental principles that this chapter is based upon is the assumption that it is impossible for large organizations to enforce standardization on a single technical architecture. We believe that this is especially true with respect to communication middleware, application platforms, or interface technology. Consequently, this chapter is not about describing specific technical standards for a service bus. Instead, we propose to look at a service bus as a kind of *meta bus*, which is comprised of the different existing software buses and middleware platforms that you will find in your organization. The job of the service bus is not only to enable basic interaction with different service components across the different platforms, but also to tie together the different higher-level infrastructure functions of these platforms.

After a general overview of the service bus concept in <u>Section 9.1</u>, <u>Section 9.2</u> looks at logging and auditing, followed by scalability and availability in <u>Section 9.3</u>, and security in <u>Section 9.4</u>.

Team LiB



Team LiB

### 9.1. Software Buses and the Service Bus

People often use the term *software bus* to refer to the technical infrastructure of the distributed environment. We consider a software bus to be analogous to the well-known concept of a hardware bus. Much as a hardware bus enables the integration of hardware parts from different vendors, for example when assembling a desktop computer, a software bus is the standardized way of hooking together any software components.

#### 9.1.1. BASIC CONCEPTS OF A REAL-WORLD SERVICE BUS

The most widely known software bus is OMG's CORBA, essentially a communication infrastructure for individual object instances. The CORBA infrastructure enables an object to locate any other object on the bus and invoke any of that object's operations. The CORBA model does not make a strict distinction between clients and servers and is essentially a symmetrical bus. CORBA is a very mature technology, but unfortunately, its underlying concept leans itself to a very fine-grained communication infrastructure that created a history of maintenance and performance problems in many projects.

Whereas CORBA is very generic bus technology with a focus on object orientation, another concept of a software bus recently emerged called the *Enterprise Service Bus* [DC2004]. Although as of this writing, it cannot be considered anywhere near a standard, its main elements are a coarse-grained XML communication protocol together with a message-oriented middleware core to perform the actual message delivery.

A number of other software buses are on the market, among them the Enterprise Java Beans as part of the J2EE specification, Microsoft's .NET, and various messaging products, including IBM MQSeries and Tibco Rendezvous. All these buses require standardization on a single interaction and communication model, as shown in <u>Figure 9-1</u>. For example, CORBA and EJB promote synchronous object-oriented communication, whereas messaging products such as MQSeries support asynchronous document-oriented communication.

## Figure 9-1. Ideal software bus that supports a single communication model. All applications must conform to the same standard, such as CORBA or MQSeries.

In a real enterprise application landscape, a single communication model will hardly suffice. Instead, you will need various communication models based on the requirements of the individual application. Vendors have long understood this point, and they provide environments that support multiple communication models at the same time. The J2EE environment, for example, supports various communication types, as shown in Figure 9-2. EJBs provide synchronous object-oriented communication, the Java Message Service (JMS) provides messaging, the system supports communication using email and SOAP, and the Servlet specification provides general support for HTTP applications.

## Figure 9-2. A software bus that supports various communication models at the same time, such as synchronous, asynchronous, or file-based communication.

In the real world, the situation is usually even more complicated because products from different vendors that support similar communication models are often in use at the same time. This situation can arise when different departments of the company introduce competing technology or when a new technology enters the environment as the result of a merger or acquisition. A typical enterprise "software bus" will usually look like that depicted in Figure 9-3.

## Figure 9-3. The infrastructure of a real-world enterprise will normally consist of various products that support similar communication models.

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 9.2. Logging and Auditing

In this book, we provide technical and organizational tools to build a system that is not prone to failures. Yet it is of course virtually impossible to rule out all failures. A lot of the measures described elsewhere in this book are potentially costly, and for this reason, they are often omitted in practice. For example, in real-world situations, the budgets for infrastructure investments to set up a failsafe database cluster or a redundant network setup simply might not be approved. Hence, you sometimes must rely on measures such as logging. <u>Chapter 8</u>, "Managing Process Integrity," has already given an in-depth discussion on process integrity for which logging is a crucial building block.

Another reason for operation disruption might be that one of the services that the application depends on must undergo unplanned maintenance. Consider an airline booking application thatamong othersrelies on a pricing service to determine the price for specific flights on a given date. Consider a serious bug within the module that performs flight price calculation. No company wants something like this to happenespecially if the calculated prices are far too lowand the corresponding service will often be shut down immediately to prevent further damage.

The bottom line is that even if you have planned for the system to handle a lot of error conditions automatically, unplanned and unrecoverable errors can still happen. We will call such errors *failures* in the remainder of the chapter. As depicted in Figure 9-9, failures require activities at different levels, including: user interaction, logging, and systems management.

## Figure 9-9. An error must be reported to the user, to a log file or database, and to a systems management system.

When coping with failures, the concepts and techniques discussed in this chapter are not only relevant to SOAs. Most of them are simply good coding practices or design patterns. However, in a distributed and loosely coupled architecture, they are of much greater importance than in a standalone monolithic application. The distributed nature of the architecture and the fact that the source code or core dumps of the actual services are not available for debugging make explicit failure-handling measures essential.

In case of such failures, it is usually necessary to perform certain manual activities to return things to normal. In lucky circumstances, this might be as easy as resetting a power switch. On the other hand, it might result in a number of employees browsing millions of rows in multiple database tables. It is not hard to see that resolving problems is a lot easier if we know where and when the error occurred and which users were involved in it. It is, therefore, mandatory that every SOA has a reliable logging and auditing infrastructure. Although logging and auditing are quite similar from a technical point of view, they differ considerably at the requirements level.

Usually, runtime output from a system is mapped to different log levels. These levels areamong other thingsused to distinguish auditing, logging, tracing, and debugging output. They are usually identified by a number of a text warnings, such as "DEBUG," "TRACE," "INFO," "WARN," "ERROR," "AUDIT," etc.

Auditing is normally put into place to satisfy some legal requirements, such as documenting that a credit card was actually charged because the client ordered flight tickets, or to document that the ticket actually did get printed and that it was sent to the address given by the client.

Auditing creates a new subsystem that by itself impacts system operation. When normal logging fails, for example, if the log file or disk partition is full, you can usually carry on merely by halting the logging process. However, if auditing itself fails, it must be considered a failure, and the system must stop its operation. After all, continuing to run without auditing in place might violate some legal obligation, and no company wants that to happen.

Tracing is usually disabled while a system is in production and is only enabled in case of a major problem because it is often so detailed that it significantly degrades system

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 9.3. Availability and Scalability

It is mandatory for any enterprise architecture to provide functionality in the way it has been commissioned to do so. In this context, we consider a system as providing availability if it is operational for 100% of planned uptime. In other words, we consider a system to be 100% available if it prevents any unplanned downtime. Note that we do not include planned downtime in our concept of availability. For example, if an application becomes unavailable because a database is taken offline for backup every night, it does not reduce the availability of the application. Instead, it limits the service level that the application supports.

Scalability in the context of this book means that an enterprise architecture provides some way to increase its capacity beyond initial planning. Capacity designates the work a system can perform in a given amount of time, commonly measured in transactions per second (TPS). Other measures of capacity are the number of concurrent users a system can support and the amount of storage space it provides. Ideally, scalability should provide a linear solution, meaning that if the capacity of the system is doubled, the resources availablememory, CPUs, network and management overheadshould at most need to be doubled. In practice, systems can scale linearly only to a given point. For most practical purposes, scalability means that a clean and defined path exists to increase the capacity of a system by a requested amount. In general, scalability is only considered up to a certain boundary. For example, a requirement might be that an application must be scalable from an initial load of 100 to 10,000 users.

One of the most common confusions that arise surrounding issues of scalability and availability is that they are often not rigorously defined or that they are defined based on insufficient information. The so-called Service Level Agreement (SLA) lies at the heart of this matter. An SLA typically defines a set of performance figures that are an integral part of the contract between one organization that provides an IT service and another that consumes that service.

The most common performance figure is the guaranteed operation time. The operation time (commonly referred to as uptime) states the time that the system must be available, along with an acceptable amount of unplanned downtime. The SLA also states the capacity of the system. This includes storage capacity, number of concurrent users, and TPS. Often, an SLA also states the response times that are acceptable for a certain percentage of requests.

Care must be taken when the requirements for any IT system are defined. Further care should be taken for any system that relies on other systems to function properly. Far too many so-called "SLAs" in the industry only state the wishful thinking of the people who originally "defined" them during a sales pitch. As a consequence, they also state requirements that far exceed what the systems actually needs to deliver in the worst-case scenario. For example, it is unacceptable to require an airline booking system to support a number of concurrent requests that is equal to all airline seats available for the entire year. Even if the booking system could fulfill this requirement, there would be no benefit for the business. Therefore, such requirements must be based on known values, such as the current number of transactions and the actual number of users that are concurrently connected to the system, where possible. If such numbers are not available, extrapolation from currently known values is required. It is both valid and commonplace to add a safety margin, but you must take into consideration that the requirements for the system's availability ultimately impacts engineering efforts. At the same time, it is valid to question whether any business process needs to operate unaffected in the cases of nuclear war or an alien invasion. Finally, you should be wary of the concept of guaranteed response times for as yet unspecified and unimplemented business functionality. It is fairly easy to plan for capacity and availability; it is almost impossible to plan for an unknown computational algorithm.

By now, it should be clear that availability and scalability come at a price. A number of applications exist in which you should be prepared to spend a large amount of money to guarantee availability. Consider air traffic control systems or a control system for a nuclear power plant. In these cases, not only the functioning of the economy but also human lives depend on these systems remaining permanently operational. In other cases, the systems that are affected by your IT system might be very expensive or very hard to replace, such as a multi-billion dollar spacecraft whose IT systems enter unplanned downtime just as it optors the atmosphere of one of the means of lupitor.

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 9.4. Securing SOAs

In SOAs, security is certainly of primary concern and should have solid foundations in the infrastructure. Before you take any technical steps to implement a specific security structure, you should capture the security requirements for all relevant pieces of software during a risk analysis. Based on this analysis, you build the software architecture to incorporate the defined security requirements. Performing a risk analysis is beyond the scope of this book, but details on the subject can be found in [PT2001]. Instead, based on a number of implementation projects carried out by the authors, we discuss here some technical issues that you must address when designing a security infrastructure in a distributed software architecture.

The main issues of a basic security infrastructure are authentication, authorization, and confidentiality. The focus broadens where the infrastructure is exposed on the outside of some well-controlled environment, such as a specific department or geographic location. In these situations, concerns such as non-repudiation, message integrity, and legally binding identity assertion become increasingly important [GS1997].

In this section, we discuss these principles and provide examples of how they relate to platforms that are commonly used to create SOAs. The guiding principle is that we want to use security as a tool to foster and encourage the use of our enterprise architecture, rather than security simply being a means to prevent incidents.

#### 9.4.1. AUTHENTICATION

Authentication means that a service caller is using a mechanism to prove its identity to the called resource. The caller must provide credentials such as a username and password or a digital certificate. Often, security requirements for authentication include an implicit authorization requirement. The typical scenario is that callers must authenticate themselves before being allowed to use the environment.

Three levels of authentication can be readily distinguished. First, it is possible for the caller to authenticate against the application frontend, second, authentication against the SOA framework is possible, and finally, authentication against a single service is also a possible option. Figure 9-16 illustrates authentication at different levels of the application.

#### Figure 9-16. Authentication against different levels of the infrastructure. In this case, the Web site supports authentication. Authentication against the infrastructure is also supported. Individual services, such as the customer service, might also require or support authentication.

[View full size image]

Authentication against the application frontend is generally straightforward and quickly deployed. Often, it will be a vital business requirement to protect access to the application itself. Authentication against individual services is often desirable in order to limit access to these services and to create a clean audit path within service invocations. However, when authentication becomes part of each individual service, a significant amount of code cluttering results. If the authentication mechanism and storage change, a significant amount of reengineering might be necessary to propagate these changes into the various services. In addition, different services probably rely on different types of storage and authentication mechanisms in the first place.

Therefore, it is generally better to insist that the user be authenticated against the infrastructure rather than against individual services. It is a prerequisite for SOA-wide authorization and identity assertion, as we discussed in the following. However, some very important advantages are available. For example, having a common user for subsequent operations in an SOA makes monitoring the environment a lot easier. Common types of attackssuch as password guessing or denial of service attackscan be prevented or at least treated in a standard way throughout the SOA. In addition, maintenance benefits greatly because changes in the underlying authentication mechanism can be performed across the board rather than against individual application frontends or services.

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 9.5. Conclusion

In this chapter, we looked at the different elements that constitute an SOA-enabling infrastructure, which we refer to as a service bus. Based on our conviction that middleware heterogeneity is an enterprise reality that can't be solved by introducing yet another set of interfaces and communication protocols, we described an approach which is based on loose coupling on the infrastructure level, effectively allowing the co-existence of multiple, heterogeneous middle ware platforms within a higher-ranking meta bus which is designed to tie together these different platforms. We also stressed the fact that this meta bus should not be developed in isolation from concrete application projects in order to avoid working on an overly abstract level. Practical experience has shown that one can work for many years on the perfect enterprise infrastructure, without ever reaching a truly stable result. Consequently, a pragmatic, project driven approach should be taken.

The different application execution containers that we find in an enterprise represent the basis for building services in an SOA. We discussed the general nature of these containers, specific implementations, and how to integrate across multiple containers.

To maximize the robustness of our SOA, we discussed dealing with system failures with the main objective of gathering as much knowledge as possible about critical parts of the service. This enables recovery from a service failure to be as seamless as possible, both automatically and manually.

To minimize such failures in the first place, we discussed concepts for scalability and availability. Both can be addressed using standard products and procedures and sometimes hardware components such as balancers are a good option. On the other hand, most frameworks for distributed objects such as EJB and CORBA support the basic building blocks needed.

Proper planning is essential for providing an available and scalable system. This holds true for the initial definition of the SLAs up to any subsequent exercise to scale a system by adding additional hardware and software.

It is fairly easy to accomplish availability at the system level and between individual requests. Guaranteeing in-request availability is considerably harder. It is not required nor appropriate for a vast number of application scenarios, and it introduces a significant overhead into system performance, very much comparable to the problems of distributed transactions. For practical reasons, the easiest way to accomplish availability is to maintain request cycles and transactions that are as short as possible. If anything goes wrong at these points, then using the strategies from the previous sectioncoping with failureswill usually be sufficient for controlled recovery.

Always consider the weak points in your application first. Keep in mind that the application is only as strong as its weakest link, both for availability and scalability.

Finally, note how much easier it becomes to cope not only with availability and scalability but also with service failures if your business logic becomes stateless or even idempotent. The strategies that deal with stateful systems are more complex and tend to impact on the overall system performance and ultimately on scalability and availability.

On top of providing reliability and availability, any IT infrastructure must support certain security features. Although technologies and tools are available that can be used to provide transparent, enterprise-wide authentication and authorization for an SOA, it is very likely that a first step will provide only authentication as part of the SOA framework itself. The reasons are mainly rooted in the fact that authentication can be introduced in most legacy systems with very little effort, while refactoring the authorization aspects of an application requires more effort. Also, it will be far easier to implement from an organizational perspective. Legacy applications are likely to be placed within trusted domains to allow for integration into the SOA without disturbing the ongoing operation of the application.

SOAs are often introduced together with a single sign-on framework. Shifting more and more aspects of application security, such as authorization, encryption, and PKI, to use the framework over time, an SOA can employ and foster the implementation of a true enterprise end-to-end security infrastructure.

#### References

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## **Chapter 10. SOA in Action**

This chapter shows how a service can be implemented in the real world to serve the needs of different usage models. You can provide the same service using different protocols and different granularity according to the context in which you use it. As is the case with traditional software, it is impossible to design a service based only on abstract principles. However, you can employ careful planning and refactoring of a service to make an implementation suitable for many different usage scenarios.

To illustrate the different design considerations applicable to different usage models, we employ a passenger check-in scenario. A passenger might check in with a mobile phone or PDA, using an electronic (or physical) ticket at an airline booth. Alternatively, the passenger might be checked in with one airline on behalf of a second airline, perhaps if the second flight leg is operated by a different carrier. In some scenarios, the same service might be accessed using multiple channels at the same time.

In its most generic form, multiple services are at work. At the heart of everything is the check-in service, assigning seats to passengers on airplanes and keeping track of the seat occupation in individual planes. As input, it takes any number of passenger ticket coupons and returns the appropriate number of boarding passes. Usually, other services will also be involved. Prior to performing the check-in, the ticket service can be used to determine which tickets are available for a passenger and to validate the tickets before check-in, in addition to marking the ticket coupons after check-in has been completed.

The customer information service might be contacted to read data regarding the preferences or frequent flyer account of the customer. Preferences for seating can be used in the booking itself. A meal preference can be used with services available from the airline's caterer to prepare only the required amount of special food, thus decreasing the airline's overhead. Personal data regarding the passenger might be forwarded to the customs or immigration office at the passenger's destination. For the time being, this discussion will be limited to the services mentioned previously, although more services might be involved for issues such as baggage handling and airport services.

As a prerequisite, it is worthwhile to determine if the provided services are aggregation-friendly. The ticket service's sole purpose is looking up tickets and checking their validity. These calls can be considered idempotent at the semantic level. If they fail, the caller can reattempt the call and expect the same result as the previous call. Invalidating the coupon is an operation that will change some persistently stored data. The call to this operation is logically tied with the assign seats call of the check-in service itself. The latter is likely to live in a local transaction and can be rolled back upon failure. In this event, there will not be any attempt to change the state of the ticket voucher, even though such changes can be easily reset in a compensating call. Finally, setting the meal preference is an operation that might also be considered idempotent. However, in an operational system, this type of process is more likely to be implemented in an asynchronous manner. In general, it is necessary only to guarantee that the caterer gets the information about meal preferences well before takeoff rather than at some specific point in time.

Throughout this chapter, we will use the same example in a set of different scenarios. In <u>Section 10.1</u>, we discuss a Web application. In <u>Section 10.2</u>, our example takes the form of an EAI integration scenario. We employ a B2B model in <u>Section 10.3</u>. In <u>Section 10.4</u>, we discuss a fat client scenario, and in <u>Section 10.5</u>, we deploy it by using a small mobile device such as a cellular telephone. Finally, in <u>Section 10.6</u>, we discuss the multi-channel scenario.

Team LiB

♦ PREVIOUS NEXT ►



Team LiB

## **10.1. Building Web Applications**

As previously discussed, Web applications are particularly suited as clients for a Service-Oriented Architecture because they offer a natural means of storing context or state information when making calls to a mainly stateless service architecture. In addition, they offer a rich user interface, and users of Web applications are accustomed to a high degree of interactivity. Although the interaction model in <u>Figure 10-1</u> remains a possibility for providing check-in functionality using a Web interface, it is likely that an airline will provide the user with a richer set of choices during the booking process.

# Figure 10-1. General service interaction for passenger check-in. The check-in service takes tickets as input parameters and creates boarding passes as output parameters, invalidating the relevant coupon of the ticket. Note that all services are basically stateless.

[View full size image]

These choices will at least include the selection of seats and meals available on the flight. On the other hand, some of the options that were originally present might not be applicable. For example, checking in online is often only possible if a registered user has purchased the ticket because logging in to the airline's Web site is required in order to perform the check-in operation. On top of that, two people traveling together must check in separately if they purchased their tickets separately.

In a Web application, users will authenticate themselves against the Web tier, usually through a Web page form with username and password or client-side certificates. Users can then be stored within the Web tier of the application. For subsequent calls to services, there are two possible models: principal propagation or trust. Principal propagation is trivial and uses frameworks that are built to support this feature. For example, within the J2EE framework, the same principal object used in the Web tier can directly be used to call other J2EE services such as Enterprise Java Beans. For other service types, such as CORBA or SOAP, the process is not as straightforward. It might include the need for mapping credentials from the Web tier to the service layer in a specific way. As mentioned in Chapter 9, "Infrastructure of the Service Bus," SOAP does not support a standardized means of passing credentials. Because Web sites can operate within a controlled corporate environment, trust between the Web tier and the service layer is a common scenario. All calls to the service layer are then carried out using a common identity or no identity at alland the actual caller principal is just passed as a parameter using call parameters. Of course, this requires that the Web application is not exposed directly to a sensitive network segment.

The interaction diagram in Figure 10-2 illustrates the need to expose the "assign seats" functionality to the outside client. In addition, displaying the available seats to the user for a given plane's seating configuration is also necessary. Although it is tempting to provide this type of layout in an ad-hoc manner within the Web application, perhaps as a number of configuration files, this implementation would soon become unmanageable. Airlines often change their seating configurations and add new planes, and it is important to provide this information to customers. In addition, the seat configuration of the airplane and the listing of available seats can be transferred within a single call, providing a good example of a coarse-grained data structure.

## Figure 10-2. Interaction that shows the check-in process for a Web application. The state of the application is maintained in the Web application.

[View full size image]

Figure 10-2 shows the full interaction diagram for the Web application invocation. Note that we made use of the results from <u>Chapter 8</u>, "<u>Process Integrity</u>," by pushing state as far up the chain as possible. In fact, all state is stored in the Web application itself. This enables a rich interaction between customer and application without the need for any stateful services. Note that the transaction boundaries of the original example have not

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## **10.2. Enterprise Application Integration**

At first glance, Enterprise Application Integration (EAI) appears to be the perfect environment to employ an SOA: In fact, plenty of reasons to use an SOA as a driver for EAI exist, and vice versa. However, EAIs pose a certain set of requirements in addition to profound non-technical issues that you must consider.

Today, most large organizations have a highly fragmented application infrastructure, in which a vast number of client applications have been created using multiple programming platforms and communication infrastructures from multiple vendors. A recent example of the endeavor to solve all application integration problems is the introduction of a corporate "standard" ERP (Enterprise Resource Planning) suite. This process failed in most organizations due to the relatively low flexibility such a solution provides and the fact that businesses have a need for tactical IT solutions, as opposed to strategically planned applications.

This example of failure provides a reason to be wary of addressing EAI problems with an SOA. Nevertheless, if an SOA is properly deployed, it will address many technical and core concerns related to EAI. There are two ways of looking at EAI in a service-oriented world: as a service provider and as a service consumer. As a service consumer, you typically want a service that is easy to access, highly available, functional complete, and stable over an indefinite period of time. As a service provider, you might want a service that is easy to deploy and to upgrade; in an EAI environment, quality of service, single sign-on, and other attributes become paramount.

At the outset, it is worth noting that so-called EAI product suites are generally unsuited to tackle the aforementioned EAI challenges. Most provide a clear integration path that is usually supported by accompanying tools, some of which provide for quality of service and ship pluggable security modules, but they usually fail to provide a stable view of the result of the integration process. Furthermore, because they only partly adhere to open or industry standards, they are very unlikely to be stable for a reasonable period of time. This is one of the prime reasons that organizations that take SOA seriously almost always choose to build their core service architecture themselves while employing standards wherever possible (see the case studies in Chapters 14 through 17).

## **10.2.1. SERVICE ENABLEMENT**

In order to provide EAI functionality in a service-oriented environment, the most common task is service enablement. Service enablement is the process that creates a service to encapsulate the functionality provided by an existing application. The type of application encapsulated can be anything from a monolithic mainframe application to a distributed application built on top of CORBA or DCOM.

Of course, service enablement is more than just wrapping and exposing an existing programming interface. As an example, consider the movement of an existing check-in application in a service-oriented domain. Assume thatas shown in Figure 10-4 the original application is a typical client/server application. As a first step for service enablement, the application is separated into a visual and a non-visual part. Communication between both parts is grouped based on their business domain. Access to the non-visual layer is defined by one or more interfaces. If possible, the implementation of the interfaces and the persistent data upon which they act can also be separated. Finally, the application is moved to the service infrastructure.

# Figure 10-4. Transformation of a monolithic application into a service-oriented application.

[View full size image]

Depending on the application, one or more services might emerge from this analysis. For example, when analyzing a real-world check-in application, it is quite likely that services such as ticketing, baggage handling, and actual check-in will surface during such an analysis. After this analysis is complete, consolidated interface descriptions for the communication can be derived. This should include provisions for undoable actions and idempotency wherever possible. The implementation and possibly the interfaces need to be

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 10.3. Business-to-Business

In a business-to-business (B2B) environment, a corporation offers some service to another company over public networks such as the Internet. In a B2B scenario, both the service consumer and the service provider benefit greatly from standard communication protocols. The service consumer profits from the increased flexibility gained when changing to another provider or when using several providers at the same time. The service provider can also profitfor example, depending on the type of communication, it might be possible to eliminate the necessity to ship software to the client.

The same goes for the security infrastructure. Because security infrastructures can get very complicated, the client and server must be certain to use a mechanism that can be trusted, where both ends of the communication chain can identify a security violation using their standard system management tools. Standards are also desirable because many interactions create a legal obligation between the parties.

In addition, it is attractive to define a standard format for the actual data being transferred. This saves both sides time and money in service development and integration. UN/CEFACT (ebXML) and RosettaNet provide examples of initiatives to establish data standards. Although the resulting standards are not yet mature and they are not appropriate for *all* B2B problems, it is worthwhile to see if they are applicable in specific cases.

 $\ensuremath{\textcircled{}}$  See the URLs list at the end of this chapter.

Most initiatives that define standard protocols result in the creation of rather large documents that are exchanged in a business transaction. This is basically analogous to a service-oriented approach with very coarse granularity. Because of the latency that can be experienced on a public network connection, it is far more efficient to send a single large document instead of making short procedural interactions.

Although the business process itself can be stateful, it pays to create stateless service invocations. In a B2B scenario, the most common way to achieve this is to pass a token along with every service invocation. These tokens can be used once or on a cyclical basis if the number of tokens is large compared to the frequency of interactions.

## Go Stateless for B2B

Stateless semantics are especially important in B2B scenarios. They enable interaction with remote customers over connections with high network latency. They also create much fewer constraints for the calling applications.

Although the authors do not believe that a runtime-naming service or repository is mandatory in an SOA environment, it is of great use in a B2B scenario. The pressure for a service to provide location transparency is far higher than in a controlled enterprise environment. After a business partner uses a service, it might require significant effort to switch to a different version. Even if it can be achieved by configuration only, it still creates an overhead that might result in unwanted downtime. Both service user and provider can be separated by large distancesas much as thousands of miles. This makes location transparency a necessity in order to provide disaster recovery capabilities. If a customer uses a service of a company whose main computing center goes down in a fire, the customer expects to be transferred to the secondary site automatically. Of course, this also includes the service repository itself being failsafe.

## Location Transparency for Stable B2B Services

B2B scenarios require real location transparency using service repositories. This enables customers to securely establish long-term relationships regardless of changes to the supplier's infrastructure.

B2B scenarios can include online billing mechanisms, although they are more common in a business-to-consumer (B2C) scenario. A B2B scenario will generally log only access information on the side of the service provider. This information can later be used to create

Page 15

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 10.4. Fat Clients

The term "fat client" is a common synonym for an application type that provides a lot of its core processes on the client machine and usually relies on a single backend system, which can be a relational database or a host system that provides some procedural capabilities.

Fat clients have a bad reputation in IT departments mainly because they created real challenges for deployment and maintenance. Because they aggregate a lot of computational logic, they require frequent rollouts of the whole application package throughout the enterprise. On the other hand, fat client applications have enjoyed a good reputation among many end users. They provide swift operations compared to many poorly built Web applications that have been created to replace them. They also provide complex interaction models and advanced graphical capabilities that are very hard to re-create using Web interfaces. Therefore, in many usage scenarios, fat clients are regarded as more user-friendly.

Although fat client applications need a considerable amount of deployment and maintenance effort, they also help to save bandwidth and increase response times compared to typical Web applications. Another strong advantage of fat clients is that they offer easy access to local hardware devices such as printers and smart card readers, which makes them particularly well suited for use in a controlled environment with medium to long release cycles.

You can use services together with fat clients to transform fat clients to rich clients, in which the client application directly accesses different services and keeps track of any necessary state information. This works in much the same way as for the Web application layer in <u>Section 10.1</u>. The core advantage is that the rich client application can be completely decoupled from the backend implementation. It thus remains robust in the face of alteration in the data model and ultimately against a possible change in the overall backend infrastructure, such as the gradual replacement of host systems with integrated services.

## **Build Rich Clients, Not Fat Clients**

Usage of thin clients does not necessarily mean reduced functionality. Services can enable you to build rich clients rather than fat clients.

Fat clients can differ in their authentication schemes. Whereas a Web application user needs to authenticate itself to the servere.g. using username and passwordfat clients might be trusted clients or might be authenticated using a certificate. Where backend systems do not support this type of authentication, the service might present some default username and password as a proxy user. When using a proxy user, you must take care to minimize the resultant security risk (see <u>Chapter 9</u>). In particular, the proxy user credential should not be stored in clear text; they must be configurable, and the allowed origins of proxy user request should be restricted to a well-defined set of network addresses.

## **Proxy Users Protect Backend Systems**

Proxy users can isolate backend systems from physical users. Because this also involves a security risk, you must take extra care to prevent the misuse of proxy users.

In the check-in example, rich clients include check-in kiosks and workstations of check-in clerks. These rich clients access printers for boarding passes and baggage tags and are usually equipped with card readers for electronic tickets, credit cards, or frequent traveler cards. Advanced check-in kiosks might also connect to baggage scales and conveyors. They provide interfaces that make it as easy as possible to navigate across the seat map of the airplane and to choose seats for check-in. Yet they do not perform the actual check-in process and do not directly manipulate data. Instead, they use a service layer that is usually identical to the one used by Web applications. Figure 10-10 shows a typical rich client setup.

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## **10.5. Designing for Small Devices**

For the foreseeable future, the design of services for small devices effectively means designing for a specific device. (For the purpose of this chapter, a small device is one with a small screen of about 150 x 100 pixels, limited processing power, memory often far less than 256kB, and limited data entry support although with a capability to create network connections to facilitate participation in an SOA.) One of the most important points is that the interaction pattern must match the capabilities of the device. Rich interaction patterns for different classes of applications often fail with small devices because of the physical characteristics of the device and also because of additional cost and longer interaction time for the user imposed by these patterns. Because network connectivity might be at a premium, the service implementation itself will usually be as lean as possible to minimize latency.

It is likely that security will be limited to mostly transport security because processing power and available memory might not allow for the encryption and signature of messages as defined in Chapter 9.

Although J2ME MIDP 2.0 supports transport security using HTTPS, there is no support in version 1.0. This means that you must use some other means to prevent passing extensive credentials between server and client. Authentications can be performed in ways such as using the telephone number, using username/password, or using certificates stored in the client. Although using the telephone number might seem like the best scenario, there are a number of reasons to avoid it.

For security reasons, runtime of environments such as J2ME MIDP do not enable access to telephony functionality, including the phone number. Furthermore, unless the delivery path is controlled, the potential for abuse of the telephone number for authentication is very high, both because it is easy to tamper with and because it is basically a public number that can be readily obtained. However, the phone number can be an excellent means of authentication when using SMS-based services, as we discuss at the end of this chapter.

One way or another, this information will be stored persistently at the client. This is mandatory due to the limited data entry support on small devices. A requirement to repeatedly enter a username and password with a clumsy keyboard obstructs usability. Alternatively, it is possible to envisage interaction without the user actually logging in to the system at all. The invocation might be triggered by a unique ID such as a reservation number.

## **Lightweight Security**

Security features often require rather large computational and memory resources. Use only security measures that have little impact on both the device and the user.

Some small devices can use SOAP (or even CORBA) for remote invocations. However, you must carefully consider whether this is a good idea. For example, SOAP requires resources at almost every level of the small device, and a significant amount of processing power and memory is used for XML parsing. In addition, SOAP libraries that are not native to the device must be bundled with the application, which often increases the size of the application beyond what is possible for the target device. When using SOAP, you must also take into account that the specification is still evolving. Because implementations for mobile devices lag back behind the current standard, often by more than a year, it is unlikely that a state-of-the-art small device can properly address state-of-the-art SOAP services in the organization.

Therefore, it is sensible to decouple the service invocation in the device from the actual service using an adapter that resides at a gateway server. This setup enables the client to connect to the service using the least common denominator in technology, usually connecting using HTTP or HTTPS. If as with MIDP 1.0only HTTP is available, the server can establish a session that prevents the repeated passing of users' credentials over the wire. The session token (or login token) can be stored in the SOAP header or can be sent using a SOAP parameter. In the authors' experience, moving from lightweight J2ME SOAP implementations, such as kSOAP, to a HTTP-POST style interaction can reduce the client

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## **10.6. Multi-Channel Applications**

SOAs are extremely effective at implementing multi-channel applications. Strengths such as building a reusable, functional infrastructure, mastering heterogeneity, and providing easy access to data and business logic are key factors for successful multi-channel projects.

A multi-channel application is characterized by a functional core that can be accessed through different channels by either human users or programs, as shown in Figure 10-15. Each channel typically has its own characteristic variant of the basic business processes and specific access technology. These specifics depend directly on requirements of the channel's users such as sales department, back office, or service center. Due to the different requirements, you will often find significant differences in the channels' business processes. For example, a sales representative will require a different view of customer data from a back office clerk or from a service center agent. The processes also depend on the access technology. As we have already discussed in the previous section, you will probably find that the processes on a mobile device with a low bandwidth connection to the functional core and a small display are different from the processes that can be supported by a Web application.

## Figure 10-15. A multi-channel application consists of a functional core that is shared by several channels. Each channel provides a specific view of core functionality, according to the requirements of the channel user and the channel technology.

Typical multi-channel applications can provide channels such as the Internet, various mobile channels such as WAP, B2B, fat clients, call centers, or voice applications. They also provide cross-channel functionality, such as co-browsing or channel switching.

## 10.6.1. FUNDAMENTAL SOA

A fundamental SOA represents the simplest model of SOA-based multi-channel applications (see <u>Chapter 6</u>, "The Architectural Roadmap"). In many cases, this simplicity makes it the preferred choice of model. In this scenario, you have services neither in the aggregation layer nor in the process layer. Consequently, you can also abandon extra tiers and the appropriate hardware, which can lead to reduced latency and improved application response times.

The authors recommend the usage of this minimal approach where possible. This advice does not apply only to multi-channel applications, either. It is beneficial for *every* SOA to keep operations as simple as possible. Abandoning complexity should be a permanent effort. Every tier that can be avoided is a valuable contribution to the system's maintainability and performance (see Figure 10-16).

#### Figure 10-16. The fundamental SOA represents a lean service approach that basically implies the usage of application frontends and basic services. No services in the aggregation layer or process layer are involved.

[View full size image]

Although a lean approach is desirable, you might find reasons to apply a more complex approach in practice. Requirements such as co-browsing or channel switching, highly complex processes, heterogeneous backend systems, load balancing, and other reasons can force the usage of façades or process-centric services.

## 10.6.2. SERVICE FA<ADE

A service façade represents a unified interface to the basic service layer for a specific project and encapsulates the functionality of the underlying services. In the airline

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

## 10.7. Conclusion

We have discussed several scenarios where an SOA can be beneficial. The requirements for the granularity that a service must provide depend on the usage scenario. The same goes for the security constraints that can be imposed upon a service. Scenarios such as a mobile one might require coarse-grained and rather insecure services, whereas scenarios such as Web-based and fat client ones will usually benefit from a somewhat smalleryet still coarsegranularity and a tighter security infrastructure. Much the same goes for the technology. Although SOAs internal to the enterprise can be based on well-understood and mature technologies, such as transaction monitors or EJBs, others such as B2B need a technology such as SOAP to offer maximum reusability and the possibility for true location transparency. Furthermore, mobile devices need the simplest protocol available to cope with the resource constraints at hand. Although it is tempting to strive for the lowest common denominator in service design, this will most likely lead to failure. It might be better to provide a service with a number of different interfaces and protocols and carefully design the required service in conjunction with the customer.

## References

[Erl04] Erl, Thomas . Service-Oriented Architectures. Prentice Hall, 2004.

[Fow02] Fowler, Martin . *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[Ha04] Harrison, Richard . Symbian OS C++ for Mobile Phones. John Wiley & Sons, 2004.

[Ri01] Riggs, Roger, Antero Taivalsaari, Jim van Peursem, Jyri Huopaniemi, Mark Patel, and Aleksi Uotila . *Programming Wireless Devices with the Java2 Platform*. Addison-Wesley Professional, 2001.

[SGG96] Shaw, Mary and David Garland . *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[BCK03] Bass, Len, Paul Clements, and Rick Kazman . *Software Architecture in Practice*. Addison-Wesley Professional, 2003.

[McO3] McGovern, James, Sameer Tyagi, Sunil Mathew, and Michael Stevens . *Java Web Service Architecture*. Morgan Kaufman, 2003.

### URLs

http://www.rosettanet.org

http://www.unece.org/cefact/

Team LiĐ

#### ♦ PREVIOUS NEXT ▶



Team LiB

# Part II: Organizational Roadmap

Part II of this book outlines the organizational roadmap to the service-enabled enterprise. We discuss the benefits of an SOA at the organizational level and look at the perspective of the individual stakeholders in SOA. Furthermore, we provide advice on how to introduce an SOA in the organization, and we provide best practices for SOA-driven project management.

Team LiB

♦ PREVIOUS NEXT ▶



# **Chapter 11. Motivation and Benefits**

Previous chapters discussed what SOAs are and provided technical details of their implementation. We now focus on the challenge of actually introducing an SOA into the enterprise. This chapter addresses the general motivation and the main benefits to be expected. Subsequent chapters discuss strategies for successfully introducing an SOA into an enterprise and for project management.

This chapter consists of two major parts. <u>Section 11.1</u> begins with a discussion on enterprise-level goals in order to round off this topic that we saw first in <u>Chapter 1</u>, "An Enterprise IT Renovation Roadmap." The pressure on organizations to become more agile and efficient is a major driving force in the introduction of an SOA, as is the inflexibility of existing IT infrastructures composed of monolithic applications. <u>Section 11.2</u> describes the viewpoints of the individual stakeholders and considers the interests of the actual people and roles involved in the endeavor of introducing an SOA.

#### Team LiB

♦ PREVIOUS NEXT ►



Team LiB

## **11.1. The Enterprise Perspective**

As described in <u>Chapter 1</u>, the main motivation for creating an SOA is the desire to increase agility of the enterprise IT systems. In addition, SOAs offer benefits at several levels, ranging from a reduction of technology dependence to a simplification of the development process to an increase in flexibility and reusability of the business infrastructure.

The ultimate goal of the additional reusability and flexibility provided by an SOA is the Agile Enterprise, in which all processes and services are completely flexible and can be rapidly created, configured, and rearranged as required by business experts without the need for technical staff (see Figure 11-1). Among other things, this facilitates a superior time-to-market for new business initiatives. This vision of an Agile Enterprise reconciles the growing demands of a global and rapidly changing business environment with the limitations of current technological and organizational infrastructures. Consequently, the Agile Enterprise is not so much characterized by a fixed state of the enterprise but rather by an ongoing change process within the enterprise.

## Figure 11-1. SOAs and their benefits.

Several sources contribute to the pressure on enterprises to instigate ongoing changes. Enterprises are continually forced to reinvent themselves by offering their customers new or better services to stay ahead of or keep pace with competitors. In order to remain as cost effective as possible, partners and suppliers must be regularly evaluated and, if need or opportunity arises, substituted by alternatives offering better quality, prices, or conditions. In addition, mergers and takeovers often require the integration of new systems, applications, and processes into the enterprise's IT and business infrastructure.

However, as unrelenting as this pressure toward constant change is, it is often obstructed by technological and organizational obstacles. For example, replacing a partner or supplier might be straightforward in theory, but it could require major efforts in reality. These efforts could involve a complete redevelopment of technical interfaces and a major redesign of business processes. Similarly, integrating a newly merged or acquired company's IT infrastructure and processes might call for a large-scale integration project. In addition, although offering new or better services might be desirable from the business perspective, it often proves to be technologically or organizationally infeasible. Typically, this can arise for two reasons:

**Too time consuming.** In some cases, developing the desired functionality might take too long to be of any use. For the introduction of a new product or service, there often is a window of opportunity (time-to-market) that must not be missed.

**Too resource intensive.** In most cases, the main risk to feasibility is the cost incurred in achieving the desired functionality. For example, replacing a supplier is only profitable if the long- and short-term costs for doing so do not exceed the savings obtained by the replacement.

For an enterprise to become an Agile Enterprise, it is therefore necessary to construct a technological and organizational infrastructure guaranteeing as much flexibility as possible. SOAs provide the means for achieving this flexibility by leveraging an appropriate architecture. In the following subsections, we describe the benefits of introducing an SOA at the level of the enterprise in detail.

## 11.1.1. AGILITY

One of the primary motivating factors for using SOAs is the potential increase of agility they offer. In order to fully understand this benefit, we must identify the different levels at which enterprise projects are threatened by *complexity*. Inevitably, complexity diminishes the enterprise's agility. The following elements of complexity can be distinguished:

**Technology.** Technical products and solutions used in enterprise projects are usually complex. Sometimes, they result from yearlong efforts to capture more and more functionality and lean towards the baroque in size and manageability. Sometimes, they are

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## **11.2. The Personal Perspective**

It should not be surprising that it is insufficient for an enterprise architecture such as an SOA "just" to be beneficial to the enterprise in order to become a success. In practice, you must also convince the individual stakeholders to support the SOA. More importantly, you must enlist the key decision makers and domain experts.

An SOA certainly can provide tremendous advantages for the individual people involved in the enterprise architecture. This section provides the most important arguments for an SOA from the perspective of the different roles in an enterprise. This information will help you "sell," the SOA to each individual in an organization, adopting each individual's role and accommodating personal requirements. In particular, we consider the following roles:

- CEO
- CIO (department for IT strategy) •
- Architect
- Project Manager •
- Functional department •
- Developer •
- Vendor of standard software (sales or product management)

#### Table 11-1. CEO

#### **Benefits**

#### **Agile Strategy**

SOA helps businesses better react to environments. IT does not limit business inevitably requires coping with some strategy, but instead enhances it.

#### Short-Term Planning

The planning horizon can be reduced drastically because SOA enables step-by-step approaches.

#### **Budget Reduction**

The budget allocated to the IT department for pure maintenance tasks can be reduced and is, thus, freed for business-oriented development projects.

#### **Technology and Vendor** Independence

The dependency of business functionality on the technological infrastructure is reduced significantly as is dependence on software vendors.

## Challenges

#### Make It Happen

Introducing an SOA means change. It resistance. A clear strategy and firm commitment are needed to overcome this resistance.

#### **Initial Overhead**

In its initial phase, the introduction of an SOA creates overhead, for which it is important to allocate a sufficient budget.

#### **ROI** Consideration

The benefit of the SOA must be quantified.

#### Reporting

Reporting to the board is a possible requirement.

#### Table 11-2. CIO

#### **Benefits**

#### Challenges

#### Independence from Technology Migration to SOA

The dependency on the underlying The existing IT infrastructure has to be migrated

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

## 11.3. Conclusion

In this chapter, we explained the motivation for an SOA introduction and the main benefits related to such an introduction. Many enterprises adopt an SOA in order to increase agility. SOAs are seen as a means of breaking up inflexible IT infrastructures, which are usually characterized by monolithic applications. The flexibility of an SOA, its modular and decoupled development process, and in particular its potential for application reuse enable enterprises to reduce their project risks and to achieve a faster time-to-market.

Obviously, SOAs are not a magic bullet, solving all problems of enterprise projects with a single strike. Only if the environment is right can an SOA yield the maximum effect. However, SOA does, if applied correctly, minimize the risks of enterprise IT by providing a sound architectural basis.

Introducing an SOA will in general be a long-lasting process, and its beneficial effects will become apparent not all at once but steadily during this process. Only at the end of the process will the Agile Enterprise become a reality.

#### References

[Sta2001] Standish Group. *Extreme Chaos*. http://www.standishgroup.com/sample\_research/PDFpages/extreme\_chaos.pdf, 2001.

#### URLs

http://www.de.cgey.com/servlet/PB/show/1004690/07.16.02%20Studie%20Web-Services \_\_G.pdf

http://www.standishgroup.com/

Team LiB

♦ PREVIOUS NEXT ►



# Chapter 12. The Organizational SOA Roadmap

This chapter describes the organizational aspects of introducing an SOA on the enterprise level. We take a close look at the political aspects of SOAs, such as the obstacles that block their successful adoption in an enterprise, and strategies and tactics to overcome these obstacles. Because every enterprise is unique, there is no universal guide to cover all eventualities in the course of an SOA introduction. However, certain patterns have emerged that are sufficiently widespread for us to discuss on a generic level. This chapter discusses these patterns and illustrates them with real-world examples of successful and unsuccessful attempts to establish enterprise-wide standards.

It should be noted that much of this chapter's content is not SOA-specific but concerns the general challenge of introducing new methodologies, or processes, at the enterprise level. The presentation in this chapter will, therefore, oscillate between SOA-specific aspects and generic aspects.

We start by providing a generic overview of the main stakeholders involved in managing an organization's IT infrastructure in <u>Section 12.1</u>. Because these stakeholders are relevant for all aspects of an IT strategy, they are also the main actors involved in the introduction of an SOA. Consequently, <u>Section 12.2</u> looks at the role that each of the different stakeholders plays in the implementation of the roadmap to the service-enabled enterprise. We discuss pillars for success in <u>Section 12.3</u>. Later, in <u>Section 12.4</u>, we describe an ideal world for the introduction of an SOA, while in <u>Section 12.5</u>, we provide some real-world examples of the introduction of SOAsdemonstrating both success and failure. In <u>Section 12.6</u>, we conclude with a series of recommendations for introducing an SOA into the enterprise.

Team LiB

♦ PREVIOUS NEXT ►



Team LiB

## 12.1. Stakeholders and Potential Conflicts of Interest

Before introducing SOAs at the enterprise level, we need to examine who the different stakeholders in the SOA are within an enterprise and the potential conflicts of interest that you must overcome to ensure a successful introduction of new technical standards and architectures at the enterprise level.

Figure 12-1 provides an IT-centric view of key roles and relationships in an enterprise. The CEO and the board of directors are responsible for high-level strategic decisions, which often will have a direct impact on IT as well. Similarly, business units drive functional requirements, and we can often observe that they present the IT department with conflicting requirements because their interests are not necessarily aligned.

### Figure 12-1. Enterprise hierarchy and potential conflicts.

[View full size image]

The CIO is often caught between these conflicting interests because investments in infrastructure are often harder to justify to business people than concrete business applications that have a measurable return on investment. CIOs thus need good arguments to defend these investmentsbecause IT is typically a cross-sectional function, it is limited by other business decisions. There is a number of major obstacles to investments in IT infrastructure such as SOA, including:

- The difficulty of providing predictable and verifiable returns of the investment that are plausible to the top management and other non-technical stakeholders
- Frequent changes in functional requirements and business strategy, which have a direct impact on the IT strategy
- Divisional interests and the mentality gap between IT and operative units
- The "not invented here syndrome" often found in IT organizations

The *return on investment* (ROI) is a major key performance indicator (KPI) for the board to approve major investments, including IT infrastructure expenses. This is typically a hard sell for three reasons:

- The return of infrastructure investments materializes in higher process efficiency and smaller future investments. However, many of today's controlling systems are not able to attribute efficiency gains to the infrastructure measures, and you can never be sure what your investments would have earned if you hadn't made the major investment.
- IT infrastructure projects have a history of unfulfilled promises, so decision makers are very critical to any kind of return calculation. For example, initiatives such as CASE, EAI, or workflow management that claimed various measurable benefits often failed to achieve them.
- Management often tends to favor short-term benefits over long-term investments.

After executives have made the most strategic decisions, it is up to the business units and the related IT projects and departments to implement systems that meet business requirements. The day-to-day interaction of business and IT people has traditionally been difficult. Business people might have a hard time understanding why technical issues are so complicated, while IT people often struggle with understanding not only certain business decisions but also the actual business itself. You can often observe a "conceptual gap" between the business and IT worlds. Typically, business requirements and the underlying technologies are extremely complex and dynamic, requiring a large number of specialists who have slightly different understandings of the environment and often differing agendas and perspectives. External consultants (strategic, business, and IT consultants) and product and service vendors with their own agendas add to this complexity. All of this increases the difficulty of matching functional requirements to a technology platform.

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

## **12.2. The Organizational SOA Roadmap**

Having introduced the architectural roadmap in the first part of this book (for example, see the different SOA expansion stages we discussed in <u>Chapter 6</u>), we will now take a closer look at the organizational aspects of the SOA roadmap. <u>Figure 12-3</u> provides a general overview of this organizational roadmap.

#### Figure 12-3. The organizational SOA roadmap.

The first step on the organizational roadmap is problem recognition. In <u>Chapter 1</u>, we provided a discussion of the reasons that lead to a phase of agony in the development of enterprise software, manifested by a decrease in development productivity and general inefficiency. If your organization is in this position, it is important to recognize this fact. You will have to determine the reasons that the IT organization is in this situation and discuss them with stakeholders before you can define a strategy for getting out of it.

Next, a number of key people have to get together and agree on the vision for the *enterprise IT renovation roadmap* and which role an SOA will play in it. Often, it can make sense to formulate a vision statement, which describes the ultimate goal, and how it should be achieved. People from both the business and technology side should be involved in formulating the vision. Although a visionary technology architect often drives such an undertaking, it is equally important that business people who can validate the concepts of the vision from the business point of view are included because they play a key role in the development processes and boards that are required to set up an SOA (see <u>Sections 12.3</u> and 12.4).

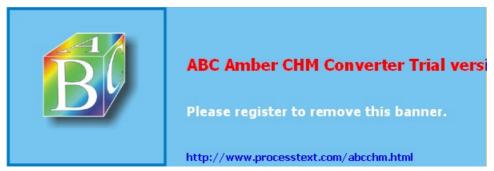
Having agreed on the vision, the next step is to define a plan that outlines how the goals defined in the vision can be achieved. This will most likely not be a detailed project plan including concrete details about implementation resources, and concrete delivery dates, but more of a high-level roadmap document that highlights the key milestones to be achieved. As we will outline in the next section, the four pillars of a successful enterprise SOA strategy include budget, backers, a team, and a project to start with. These should be included in the plan.

The development of this plan will most likely go hand in hand with the decision making process, which will eventually lead to the go/no-go decision. Assuming a decision for the execution of the plan is made, the typical next step is to start with a suitable pilot project, as identified in the plan. The next section will provide more insights into the ideal characteristics of this pilot.

Finally, it is important to realize that in order to successfully establish an SOA on the enterprise level, you must constantly keep track of the project's status in order to fine-tune and steer the overall strategy. The introduction of an SOA is not a once-off project but instead requires constant efforts to ensure that future development projects will adhere to the architectural principles of the SOA. As we discussed in <u>Chapter 1</u>, enterprise architects constantly must fight the battle between tactical, short-term development and strategic refactoring and architectural compliance (see <u>Section 1.3</u> for more details). Thus, the enterprise-wide rollout of the SOA should really be seen as an activity that runs in parallel to the day-to-day project business of the IT organization, including as much motivation work as technical guidance.

Team LiB

♦ PREVIOUS NEXT ►



Team LiB

## **12.3. Four Pillars for Success**

Although a wide variety of factors determines the success of an enterprise's SOA strategy, four main pillars for success can be highlighted: securing a budget, choosing a suitable project to start with, setting up a strong SOA team, and finding backers and buddies (see Figure 12-4).

# Figure 12-4. Four pillars of an SOA's success: budget, project, team, and buddies.

## 12.3.1. BUDGET

Obviously, *securing a budget* is a sine qua non for any successful introduction of new technology within an enterprise. For one thing, this budget will be needed to *finance one or more initial projects* acting as pilot applications of the SOA. Because the success of these projects will have a considerable impact on the (perceived) success of the overall SOA introduction, they should be chosen with great care, as we will explain in detail later. It is also crucial that they are equipped with sufficient budget to meet the challenges inherent in the use of new technologies, methodologies, and processes.

In addition, a budget is needed to *compensate for initial overheads* caused by applying an SOA. Such overheads are caused by different factors. For one thing, employees have to familiarize themselves with new standards and processes. More important, however, is the initial overhead caused by efforts required to increase reusability. Instead of merely focusing on immediate requirements, potential future applications must be taken into account to ensure that the implemented service is reusable.

Even if a business department is supportive of an SOA in principle, it might have problems providing the resources needed to account for this overhead. In such a case, it is good to have a budget available to cover the necessary resources.

## **12.3.2. INITIAL PROJECT**

The second pillar is the *choice of a suitable project* piloting the implementation of the enterprise SOA.<sup>(1)</sup> You must take into account several criteria to determine good candidates. First, the chosen project should be as *visible* as possible. This does not necessarily mean that it has to be particularly prestigious. On the contrary, choosing a less prestigious project might be advantageous because it diminishes the immediate pressure during the project's realization. However, the functionality provided by the implemented services should be widely used in the enterprise. On one hand, this ensures that the results achieved in the project will be highly visible. On the other hand, it will guarantee a significant reuse potential of the implemented services, which in turn will contribute to the validation of the benefits of the SOA and will help to sell it.

<sup>10</sup> Obviously, more than one project might be chosen to pilot the SOA. For ease of presentation, our discussion will focus on a single pilot project.

Ideally, the project should run no longer than two or three years, with a first delivery after six months. There should be a clear technological scope based on equally clear business requirements. In fact, it is crucial that the project have a business focus with measurable benefits instead of just being technology-driven. This not only concerns the project as a whole but also holds true for the individual services developed in the project. The more obvious the benefit of these services, the easier it will be to prove the SOA's ROI.

It is also a good idea to carefully evaluate the business department that will be responsible for the realization of the pilot project. Ideally, it should be enthusiastic toward the SOA idea, but at the very least, it should be open and positively biased. Otherwise, you risk too much friction during the delivery of the pilot scheme, which might subsequently jeopardize the entire SOA endeavor.

## 12.3.3. SOA TEAM

The third pillar is setting up a special SOA team. Such a team should focus exclusively on

#### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 12.4. An Ideal World

In the previous section, we discussed four pillars for ensuring the successful introduction of an enterprise's SOA. In this section, we will describe in more detail those structures and processes that should be established to achieve success. In doing so, we will draw the rosy picture of an ideal world. Subsequent sections will deal with the intricacies you will encounter in real-world scenarios.

#### **12.4.1. STRUCTURES AND PROCESSES**

A number of building blocks are useful for the successful introduction of any new enterprise-wide technology or methodology, namely:

- Whitepapers
- SOA board
- Multi-project management
- Standard processes
- Support of all actors

The following paragraphs will describe these generic building blocks in more detail. We will then address issues that are specific to the introduction of an SOA.

*Whitepapers* are a good medium for describing basic principles and should be used to manage the "why" and "how" issues. Ideally, there should be several whitepapers, each dealing with a particular aspect of the SOA (see Figure 12-5). A *strategy whitepaper* should explain the overall goal of the enterprise's SOA and its perspective over the next three to five years. Aspects to be addressed include, for example, integration with the existing infrastructure, the main business drivers behind the architecture, and the potential for integration across enterprise boundaries, such as with suppliers, partners, and customers.

#### Figure 12-5. Whitepapers must address various target groups.

A *business whitepaper* should focus on the business benefits expected from the introduction of an SOA. Ideally, it should contain a business case including predictions concerning the ROI. It should at least demonstrate the benefits of an increased reuse of implemented business functionality and the potential for the efficient development of new services for customers, employees, and partners. It could also highlight those aspects of business functionality that are ideally suited to reusability.

Finally, a *technology whitepaper* should address the technological issues involved in implementing the SOA. On one hand, it should explain in detail how integration of the SOA with the existing technological infrastructure is envisaged, in particular concerning issues such as asynchronous messaging or transactional behavior. On the other hand, it should describe details of the technological realization of the SOA itself. In many cases, a special platform will be used or developed to realize the technical infrastructure of the SOAthe service busand a technical whitepaper is a good place to specify the scope of such a platform and a roadmap for its implementation.

The repository is one of the key ingredients of an SOA, and it will be highly visible when services are available. The technical whitepaper should describe the repository structure in addition to processes for using and enhancing it.

Whitepapers are a good starting point for disseminating information about a new technology or methodology in an enterprise. However, as they are merely papers, their power is rather limited. What is definitely needed is an organizational entity responsible for making a technological vision work in everyday life. One way of achieving this is to establish a dedicated *SOA board*, which is responsible for the promotion of the SOA idea and the monitoring of its application in actual projects (see Figure 12-6).

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 12.5. The Real WorldOrganization-Wide Standards

This section presents some examples from the real world. We begin with an example of the failed introduction of a platform project, and afterwards, we summarize the positive aspects of two of the use cases presented in detail in <u>Part III</u> of this book. We conclude with a summary of the lessons learned from these real-word examples and a brief sketch as to how to deal with standards spanning several enterprises.

#### 12.5.1. AN EXAMPLE OF FAILURE

The example in this subsection is based on real-world experiences in a platform development project of a large corporation. We will use the arbitrary acronym COLOSS in the following to refer to this project and the platform itself. COLOSS is realistic example of a failed introduction of a new technology and offers instructive insights into the challenges and pitfalls of such an undertaking.

<sup>21</sup> There is definitely no connection to the EU project in the 5th framework bearing the same name, or any other projects named COLOSS that we are not aware of.

We would like to point out that although the following description is based on a single project, it is in no way unique. Many of the problems described here are typical for large-scale projects introducing or applying new innovative technology.

In fact, we expect many readers to immediately notice parallels between COLOSS and their own experiences with similar projects.

The main purpose of the COLOSS platform was to provide host functionality in a standardized way, such that arbitrary frontend and mid-tier applications could easily use this functionality. The following is a brief description of the results:

**Launch postponed.** Initially, the project was supposed to deliver a first version of the platform after three years. This is already a considerable time frame, even for a strategic development project. However, when the platform was not "finished" after the initial three-year period, its launch was postponed several times, until after five years it became apparent that the whole endeavor was seriously flawed.

**Scope creep.** During the project, more and more functionality was assigned to or claimed by the platform. This particularly included functionality for session management, security, and transactional behavior.

**Obsolete technology.** Due to the long project duration and the additional delay, the technology used in the project became more and more obsolete.

As a consequence, support for the COLOSS platform crumbled. Whereas in the beginning, the platform was envisaged as a significant step forward that would considerably facilitate development of new applications, it was seen as more and more of a bottleneck threatening all development projects in the enterprise.

For example, projects began to devise workarounds that would access host functionality the "traditional" way because it was unclear whether COLOSS could provide the corresponding functionality on time. Similarly, projects developed their own solution for platform functionality such as transactions, security, session management, etc.

Instead of standardizing and facilitating the enterprise-wide development process and fostering reuse, the failure of the COLOSS project caused an even more heterogeneous infrastructure with many redundancies.

In hindsight, several lessons can be learned from this failure. You should bear in mind the following key recommendations when introducing a platform based on new technology:

**Avoid Technology Focus.** Perhaps the most critical mistake of the COLOSS project was that it was conceived as a technical platform development project that was not immediately tied to any business project. Though this made sense from a conceptual viewpoint, the IT focus caused a lack of synchronization between IT and business projects and was also ultimately responsible for scope creep and the delayed launch.

**Start Small.** Instead of aiming at a fully developed platform providing a high degree of functionality, it would have been more reasonable to start with a small prototype offering limited functionality. First, such a prototype would have been finished after a smaller time

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## **12.6. Recommendations for the SOA Protagonist**

This section contains major recommendations for the SOA protagonist regarding politics and tactics of SOA introduction.

**Solid foundation.** We identified four main pillars for success, namely securing a budget, selecting adequate pilot projects, building a team, and finding backers and buddies.

**Establish processes, structures, and standards.** In order to make sure that the SOA is not just a nice concept written on paper, processes, structures, and standards must be established. It is well known that there is a major difference between theory and practice. Even if employees or departments are fully supportive of the SOA idea, they might disregard its principles in their day-to-day routines for several reasons, such as a lack of resources to cope with the overhead caused by the SOA, a lack of support for the project, or simply the reluctance to familiarize oneself with the details. It is vital to make sure that the SOA principles are not just laid down on paper but are actually applied in daily practice.

**Enforce substantial change.** In some cases, new methodologies and technologies are introduced in an enterprise without any real change happening. Everything remains the same, except that new labels are used to describe what is done"Yes, of course, all the functionality developed in our application projects is now service-oriented." For an SOA to become effective and more than a void bubble, it must contain substance. Only if it has an impact on existing structures and processes, such as by transforming or extending them, will it yield benefits.

**Ensure business involvement.** Although SOAs are not primarily about technology, technology issues can become too dominant in the SOA introduction, especially if a dedicated platform is to be developed as a basis for the SOA. Ensuring business involvement is therefore crucial from the very beginning. Projects should be driven by business needs and should yield measurable business benefits.

**Focus.** It is important to have a clear focus when introducing an SOA and to concentrate on feasible achievement and reasonable time frames. This ensures early visibility of benefits and minimizes the risks of individual projects failing and thereby endangering the overall SOA introduction.

**Evangelize.** The SOA introductions should be permanently accompanied by evangelistic activities, such as coaching, training, education programs, and whitepapers.

**Cope with open or concealed opposition.** Inevitably, not everyone in the enterprise will be thrilled by the prospect of an SOA. Single employees or whole departments might openly or secretly oppose the introduction of an SOA. Reasons for such opposition can be manifold. They could be rooted in a general dread of change regardless of its concrete nature, or they could be related to a specific fear of losing influence and/or control. It is important to constantly watch for signs of open or concealed opposition and deal with it adequately. In this context, it is extremely important to precisely understand the motivation of the other stakeholders and provide offerings that are actually perceived positively. If the fear of coping with change is the greatest concern, coaching or training can mitigate the opposition. If the key problem is about losing influence, it could also be helpful to integrate people into the SOA processes and give them appropriate responsibility to contribute to the SOA success.

**Compensate overhead.** A particular aspect that is easily overlooked is the fact that applying an SOA will create an initial overhead. This overhead must be taken into account in the budget.

**Ensure visibility.** In order to firmly entrench the SOA in the enterprise, high visibility should be ensured, such as by involving all relevant actors in the processes and by implementing widely used functionality as services in the pilot projects.

Team LiB

♦ PREVIOUS NEXT ►



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

◀ PREVIOUS NEXT ►

## 12.7. Conclusion

In this chapter, we examined the political and strategic aspects of an SOA introduction. We pointed out that introducing an SOA is a complex endeavor that can only succeed if it is handled professionally and with adequate focus. It usually takes several years before an SOA is really established within an enterprise.

The real-world examples have illustrated the most common challenges encountered when introducing an SOA and some suitable methods to successfully deal with them. However, SOAs address the concurrent trend towards aligning IT with overall business goals. The service-enabled enterprise facilitates more efficient SLAs between IT and the business organization, and thus IT is increasingly "being brought into the fold."

#### URLs

http://www.isixsigma.com

http://www.iso.ch

Team LiĐ

♦ PREVIOUS NEXT ►



# Chapter 13. SOA-Driven Project Management

Modern project management methodologies have an interesting and eventful history. One of the earliest projects that adopted rigorous project management processes was the Manhattan Project in the 1940s, in which the United States developed the first nuclear weapon, a massive research and engineering undertaking [Gro83]. In the 1970s, practitioners in industries such as defense and construction started to adopt project management methodologies. The 1990s saw a migration toward project management methodologies, starting with TQM in the mid-80s, process reengineering in the early '90s, risk management and project offices in the late '90s, and the currently ongoing wave of mergers, acquisitions, and global projects of the new century.

Some of the generic project management practices and tools are directly applicable to software development. Gantt charts and network diagrams are frequently used not only in construction industry projects but also in software projects. Generic project management standards such as PRINCE 2 address organization, plans, controls, stages, risk management, and quality, configuration, and change control, all of which also apply to any software project. Today, a wide variety of project management methodologies address the specifics of software development projects, ranging from the simple and widely used waterfall model to sophisticated, iterative models such as Rational Unified Process or Catalysis.

As in the remainder of this book, this chapter is based on the assumption that the projects we are looking at are related to enterprise architectures, including packaged applications and bespoke software, with complex dependencies and integration requirements.

In this chapter, we limit the discussion of generic software development project management methodologies to a brief introduction. Our focus is on understanding how aspects of SOA can be used in the context of some of these established project management methodologies in a complementary manner. We introduce the concept of SOA-driven project management, configuration management, and testing from the perspective of the project manager.

#### Team LiB

♦ PREVIOUS NEXT ►



# ABC Amber CHM Converter Trial versi

Please register to remove this banner.

Team LiB

## **13.1. Established Project Management Methodologies**

Like any engineering, manufacturing, or product development project, a software project must cope with the conflicting requirements of time, resources, and functionality. However, software projects are often somewhat special in that they exhibit a number of problems not normally found in other industries. In particular, enterprise software is tightly coupled with the internal organization, processes, and business model of the enterprise, as we discussed in <u>Chapter 1</u>. Naturally, this means that the interfaces between enterprise software and human users have a much higher complexity than interfaces between human users and other types of complex technologies.

Take, for example, a sports car with an extremely sophisticated engine, anti-sliding system, and exhaust reduction systemat the end of the day, the interfaces exposed to the user are relatively simple: the user controls the technology through the use of the steering wheel and brakes and is typically not aware of the highly complex technology hidden under the hood.

Compare this, for example, to a software system such as a CRM system or an ERP package. Such a system requires much more complex user interfaces, and interaction with such a software package is much more direct and frequent in the day-to-day business of the end user. Thus, enterprise software projects usually require much tighter interaction with customers during the development phaseafter all, it is the customer's business that we are modeling one-to-one, with all the corresponding details, day-to-day operational processes, and special cases.

Unfortunately, enterprise software projects very often suffer from the *I can't tell you what I want, but I will recognize it when I see it* phenomenon. Early project management methodologies that were developed specifically for software development projects where not able to cope with this issue. Most notably, the popular waterfall development model implicitly assumes that customer requirements are fixed at the beginning of the projects and that changes to these requirements are the exception, not the norm. The phases of the waterfall model include requirements specification, high-level and detailed design, coding, module and integration testing, and acceptance testing. The waterfall model is based on full documentation at the completion of each phase, which must be formally approved before the next phase is entered. The final delivery is one monolithic result.

Because of the limitations of this approachin particular the inability of the waterfall model to cope with unstable requirementsa number of more evolutionary approaches for software project management have emerged over the past 20 years. These more incremental or iterative models are built on a philosophy of interaction and change, which is much better suited for coping with unstable functional requirements. These models are typically based on frequent releases, which are used as the basis for getting early and continuous customer feedback. Instead of delivering the first work result after months or even years (as in the waterfall model), the iterations of these evolutionary approaches typically last only a few weeks (or even days).

An early representative of these development processes was James Martin's *Rapid Application Development* (RAD), which is based on incremental stages. Each increment represents a short-duration waterfall. Barry Boehm developed one of the first fully evolutionary models, widely known as the *Spiral Model*, which is based on a set of full development cycles that continually refine the knowledge about the final product and help control project risks.

A number of very complex and sophisticated development process models have been commercialized in the past decade. Most of these models are based on an iterative approach in combination with some form of object-orientation, such as using UML as a formal modeling language. Probably the most prominent representative of this class of development processes is the *Rational Unified Process* (RUP), which is now owned by IBM. RUP is iterative, depends heavily on visualization through UML, and uses component-based architectures. In the same arena of relatively heavyweight iterative development processes, we would also find *Dynamic Systems Development Method* (DSDM), *Microsoft Solution Framework* (MSF), and *Catalysis*.

In the wake of the fast-moving Internet revolution of the late 1990s, a number of approaches emerged that took a much more lightweight approach to iterative software development, often referred to as *agile* development. In 2001, several representatives of

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## **13.2. SOA-Driven Project Management**

As we said in the introduction, this chapter focuses on how service orientation can support project management without inventing an entirely new project management methodology. Naturally, the level which SOA elements should be included in project management depends strongly on the expansion sta the SOAan organization that is already further down the road in terms of rolling out the SOA will in som cases be able to benefit more from these concepts. However, even in the early stages, an organization benefit greatly from the concepts outlined in this chapter.

When looking at SOA-driven project management, it is important to recall that an SOA introduction hap on many different levels within an enterprise:

**Business projects versus IT projects.** First of all, any SOA-driven project management will have to be closely aligned with concurrently ongoing business projects, which are the source for any functional requirements. A general theme throughout this book has been the close relationship of the services developed for our SOA with concrete business functions. As outlined in <u>Chapter 1</u> and consecutive chapt the services in an SOA are often a one-to-one mapping of a business entity such as a process or a transaction. Thus, services are an ideal tool for coordinating business projects and IT projects, giving promanagers from both sides a perfect means for communicating and aligning business requirements and technical implementation. Often, we find that multiple business projects will have an impact on an SOA project and vice versa.

**IT program versus IT project management.** Next, on the IT level, we need to differentiate between management of individual IT projects and the management of multiple IT projects (program management in Section 12.4.1, we introduced the concept of an *SOA board* as a key tool for coordinating multiple print a program. Section 12.2 provided an *organizational roadmap* and discussed how the different stakeh and influencers must be included on the program management level. Section 12.2.1 describes how SO artifacts can be used to control individual projects and sub-projects within them, as well as to coordinating multiple projects on the program management level.

**Business services versus SOA infrastructure.** Finally, it is important to remember that an SOA introduction has two architectural levels: the actual business services themselves and the required services infrastructure, which enables different services and service consumers to interact with each other in controlled, secure, and reliable way. <u>Chapter 6</u> outlined the different expansion stages of an SOA, inclu fundamental, networked, and process-enabled SOAthe level to which an SOA can be leveraged for project management purposes will depend on the expansion stage that the SOA has reached in the enterprise. are in the early stages of SOA development, recall our suggestions in <u>Section 12.5.1</u>: Start small and a technical focus. In particular, if you are in the early stages of putting your SOA infrastructure in place, a putting too much functionality into the initial platform. In addition, don't develop the platform on its or instead make sure that it is developed within the context of a concrete project, which ideally adds signibusiness value. <u>Chapter 9</u> introduced the concept of a "meta service bus," which can cater for adoption different technologies in an evolutionary way. <u>Chapter 14</u> discusses a concrete case study from Credit S outlining how the company introduced a synchronous information bus, an asynchronous event bus, and transfer-based integration infrastructure driven by the demand from different projects, which in turn we driven by concrete business demands.

As we will see, an SOA can help cut projects into more manageable pieces, which helps program and p managers to coordinate concurrently ongoing business projects, IT application projects, and IT infrastruprojects. Figure 13-2 again highlights the different levels of dependencies that an SOA can help to coordinate.

# Figure 13-2. SOA-driven program and project management contributes to the coordination of business and IT projects. It also enables a stepwise extension of business infrastructure (deployed services) and the technical infrastructure (see bus).

#### 13.2.1. USE SOA ARTIFACTS AS PROJECT CONTROL ELEMENTS

A key issue in software project management has always been the mapping of project control elements as tasks, work breakdown structures, etc.) and software artifacts (program code, data models, specifica and the complex relationships between all of these). Page 195

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## **13.3. Configuration Management**

Configuration management in an SOA project requires an approach that is somewhat different from usual practice. Traditionally, each project creates a single repository in configuration management systems such as CVS, Rational Clear Case, or Telelogic's Continuus. Such an approach is not practical in an SOA for the following reasons:

- Services often do not belong to a single project.
- Service infrastructure is used across all participants of the SOA.
- The SOA should enable the independent deployment of individual services.
- The access to the source code of individual services must be controlled independently.

We discuss these issues in great detail in the next section along with some proposed solutions.

#### **13.3.1. CHALLENGES FOR AN SOA CONFIGURATION MANAGEMENT**

In an SOA, not all artifacts generated by a project will ultimately be owned by this project. Instead, the services that are intended to be reused in other projects will be owned by the organization. This is necessary due to the mode of reuse that one strives for with SOA.

Traditionally, reuse has been achieved by either reusing source code or sharing libraries between different applications. This will lead either to transfer of ownership of the copied code fragments to the new project or to tying the project to a certain version of a library that has been used. SOA, on the other hand, focuses on the reuse of software components at runtime, effectively reusing existing business systems including the life data they own. This creates a set of dependencies completely different from those of the reuse of code or libraries. Reuse of existing services will raise the need to amend these services or to fix errors within these services that are only discovered in subsequent reuse. At the same time, a project can be expected to come up with some services that will in turn be made available to the entire enterprise (see Figure 13-11).

# Figure 13-11. An SOA leverages a scenario in which multiple projects typically share common services.

Much the same holds true for certain support code that is written for a number of specific services, regardless of the eventual ownership of these services. Examples include logging components (see Chapter 9) and transaction handling (see Chapter 8).

It seems beneficial to be able to maintain, build, release, and deploy all shared services and to some extent the supporting code independently from each other. Otherwise, the agility that the SOA approach enables might be undermined by the requirements of the release and deployment process.

There is no apparent reason why independent services that are created during any particular project should only be deployable and maintainable together. In fact it seems largely beneficial to separate them as much as possible. Consider a service that provides customer-related information within an airline corporation. This service might have been created originally to support booking services during a booking project. As a typical cross-corporate service, it can be reused by other projects. All requested amendments apply to the customer service but not the booking application and its booking services. Ownership of the customer service itself might at some point actually move into another project, for example one that supports a customer retention program. Here, the customer service will be developed and deployed totally detached from its originthe booking application.

#### 13.3.2. RECOMMENDATIONS FOR THE SOA INTEGRATION TEAM

Although the creation of an appropriate structure for configuration management (CM) is a

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 13.4. Testing

Testing is probably the major quality control tool in any software development. The term testing in this context refers to systematic, automated, reproducible testing, rather than the ad-hoc testing approach that is still dominant in many software development efforts. This formal approach generates objective and measurable test results that can be used to obtain a measurement of the quality of the created software artifact.

Testing is best grouped into different categories, depending on the required objective and level of granularity. First, load testing and functional testing must be distinguished.

Load testing means testing a component under a specific load for a defined time. It is crucial to judge whether the software can meet any required SLAs. Load testing normally requires that the test be conducted against an environment where all the backend systems of the component are available and perform and scale as they will in the live environment. Otherwise, the response times or stability numbers don't mean much. For example, if a test is carried out against a simulation of a message queueing system, there is no knowing if systematic failures of the actual system will keep the performance of the testing component within the required range.

Functional testing means ensuring that the operational results of a software component are consistent with expectations. Functional tests that execute a single call with a given set of parameters and that then compare the result of these calls with the expected result are referred to as unit tests. Several unit tests can be chained into a testing series, testing several related and possibly sequential actions. In addition, test robots can automate tests of an entire application frontend by simulating user actions, again comparing results with expectation. Automated test tools can execute thousands of tests in short periods of time, usually far more than can be done manually. This special form of chained unit testing is commonly known as an end-to-end functional test. When a single componentsuch as an individual serviceis tested, functional testing might well allow for a certain part of the application to be simulated. For example, persistence using an object relational mapping library can be replaced using a simulation of the library. The upside of this approach is that database setup scripts and resources need not be available and initialized at time of testing, reducing testing time and speeding the quality assurance process. In contrast, when a component is functionally tested with all its backend components available, this is referred to integration testing for this component.

Of course, some overlap exists between the test types because load test tools often provide some mechanism for result checking and unit test tools provide some mechanism for generating increased load. Still, the test scenarios described remain different because they address different problems, often at different stages in the development lifecycle.

Systematic testing, in particular functional development time testing, has become widely popular with the advent of agile development methodologies such as extreme programming. However, it often poses a non-trivial problemdeciding which of the created artifacts justifies creation of a dedicated test. Test design is by no means easy because any functional test must be reproducible and must achieve as much coverage as possible. The danger of "testing the obvious" is real, and even large test sets have limited value if they break the first time the components are called with unexpected parameters. In addition, building tests is development work in its own right and might require building dedicated software components, for example a simulation of a backend or initialization scripts for databases. Still, tests must be as simple as possible to avoid the need to create a "test for the test."

The nature of SOAs can facilitate finding the most important functional test cases. Mission-critical enterprise applications might be rendered useless if one of the service components stops functioning properly after a new release. For this reason, the service component itself is the prime candidate for functional, integration, and load testing. This does not mean that end-to-end testing or testing of single libraries will no longer be required. It merely dedicates a large portion of the testing effort to testing services.

Consider the example in Figure 13-14, which shows a customer retention service that is composed from multiple services. Two of these services are shown in the figure: a printing service and a service that provides basic customer data. The customer retention service has multiple clients, among them a browser-based call center application that supports telephone marketing to the existing customer base and a number of batch programs that

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

### 13.5. Conclusion

In this chapter, we presented best practices for SOA projects. Most importantly, these practices do not represent or require a new project management methodology. SOA-driven project management means adopting a set of useful, SOA-specific practices that are complementary to an established methodology.

SOA project management starts in the first minute of a new project. A first draft of the high-level service design is a major deliverable from the project definition phase. At the core of SOA-driven project management, we find SOA artifactsin particular, service contracts and services, which we leverage as project control elements. Most important, SOA-driven project management enables the efficient decomposition of complex, potentially heterogeneous distributed systems into manageable sub-systems and the disentanglement of the dependencies between them. If used properly on the project management level, *service iterations* are the right tool for managing the *heartbeat* of the project.

Furthermore, SOAs enable enterprises to put an efficient configuration management in place. Nevertheless, configuration management is regarded as a highly complex task, reflecting today's heterogeneous enterprise reality.

Finally, we described service-driven regression testing as another key factor in SOA success. The particular service design enables efficient testing of enterprise applications, that is, the encapsulation of services, their distinguished business meanings, and the clearly defined, coarse-grained interfaces.

#### References

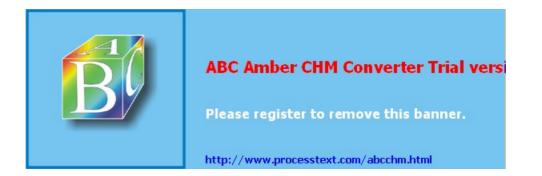
[Gro83] Groves, Leslie R . Now It Can Be Told: The Story of the Manhattan Project. Da Capo Press, 1983.

#### URLs

http://www.agilemanifesto.org

Team LiB

♦ PREVIOUS NEXT ▶



#### Team LiB

## **Part III: Real-World Experience**

The third part of this book describes four cases of successful SOA introductions on the enterprise level, looking at them from the business and the technical perspective. The case studies include Deutsche Post, Credit Suisse, Winterthur Insurance, and Halifax Bank of Scotland.

#### Team LiB

♦ PREVIOUS NEXT ▶



## **Chapter 14. Deutsche Post AG Case Study**

This chapter describes the introduction of a Service-Oriented Architecture at Deutsche Post. Deutsche Post belongs to Deutsche Post World Net, a multinational group with more than 275,000 employees, comprising also the brands DHL and Postbank. The SOA described in this chapter was set up for the MAIL Corporate division at Deutsche Post, a partner to three million business customers, providing services to 39 million households through 81,000 delivery staff, 13,000 retail outlets, 3,500 delivery bases and 140,000 letterboxes. Deutsche Post is Europe's leading letter mail service provider and the number one direct marketing company in the German market. Currently, the SOA is also rolled out at DHL, which counts 50 percent of the "Forbes 500" companies that have logistics requirements among its customer base. DHL has a global presence in 150 countries, a total of 430 terminals/warehouses and a total of 45,000 employees.

Deutsche Post's decision to introduce a Service-Oriented Architecture was based on several considerations. Deutsche Post's IT landscape grew significantly for the last years. Such a huge, distributed and complex infrastructure is not easy to maintain especially concerning the core business processes. In addition, the development of new applications became difficult. Numerous applications were so-called island solutions instead of holistic, business-driven solutions. Moreover, applications did not have clear functional boundaries, which led to considerable functional redundancy between applications and made modifications complex and resource intensive.

Finally, the maintenance of the IT architecture used up a considerable amount of the overall IT budget and offered hardly any access to core information about revenues, cost, and competitor information, which is crucial in today's dynamic business environment. This information was scattered over many components of the IT landscape and had to be consolidated via complex processes. The need for a consistent and centralized data storage became apparent.

Given this situation, Deutsche Post decided to introduce a business-driven SOA. In the initial concept of this SOA was produced in 1999, and the actual implementation started in 2000. In addition to business services, a service infrastructure, the Service Backbone (SBB), was also realized. This backbone was launched in December 2001 and has since then been successfully used as the technical basis for Deutsche Post's SOA.

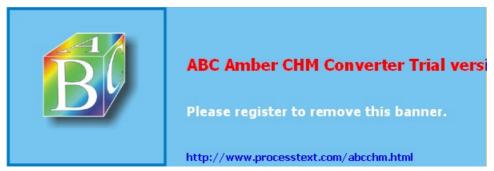
In The SOA is called Business Domain Model (BDM) at Deutsche Post.

As we have indicated in <u>Chapter 12</u>, a great deal of an SOA's success depends on properly defined processes and comprehensive documentation. Deutsche Post therefore provided a set of three articles defining the foundation of their SOA (see [<u>HBa04</u>], [<u>HBr03</u>], [<u>HS03</u>]). This case study makes extensive use of this worthwhile documentation.

This chapter describes in some detail how the SOA has been implemented at Deutsche Post. In <u>Section 14.1</u>, the general scope of Deutsche Post's architecture is presented, both from a business and technical perspective. <u>Section 14.2</u> then discusses the organizational structures and processes used to implement the SOA. The technological perspective is presented in detail in <u>Section 14.3</u>. Finally, <u>Section 14.4</u> describes the lessons learned, the benefits achieved by Deutsche Post, and future perspectives.

#### Team LiB

▲ PREVIOUS NEXT ▶



Team LiB

## 14.1. Project Scope

Deutsche Post sees its SOA as a business-oriented IT strategy. The main objective is to standardize access to functionality and to ensure reusability. The SBB (Service Backbone)the major outcome of the infrastructure developmentis described in detail here.

When starting to restructure its IT landscape, Deutsche Post followed the approach summarized in Figure 14-1.

# Figure 14-1. Deutsche Post's use of a Business Domain Model to get from business processes to IT applications.

[View full size image]

The first step was the creation of a business requirement evaluation, which led to the redesign of many business processes according to the analyzed input-output relations. The Business Domain Model (BDM) that was designed in further process steps is comprised of modular components. Closely related functionalities and data are bundled in domains.

<sup>22</sup> A brief remark regarding terminology: The domains at Deutsche Post correspond to what is called "services" in this book, they provide a cluster of functionality. The service implementations at Deutsche Post correspond to what is called "interface" in this book.

#### 14.1.1. BUSINESS IMPACT

One of the most important benefits of Deutsche Post's BDM is the fact that it enables a view of IT applications from the business perspective by providing appropriate representation of business-oriented components, their interfaces, and interconnections.

Deutsche Post utilizes an insightful metaphor to promote its SOA internally (see [HBa04]). Deutsche Post compares the concept of its SOA to a map of a city. BDM's high-level components are called domains. They describe reasonably separated components, which contain the main business logic. Deutsche Post compares these domains to different districts of a city. Every district (such as airport, residential area, industry parks, etc.) has a clearly defined purpose. The urban infrastructure (such as streets, and electricity and water supply) connects the different districts to each other and is compared to Deutsche Post's Service Backbone, which provides access to the business logic encapsulated by the domains.

Figure 14-2 shows the Business Domain Model used at Deutsche Post and its use of the domain concept. As the main components of this construct, domains contain modularly defined functionality and thus enable the support of business processes and the underlying data.

#### Figure 14-2. Deutsche Post's Business Domain Model with domains.

[View full size image]

According to Deutsche Post's motivation paper [<u>HBa04</u>], the main characteristics of domains are as follows:

- Domains encapsulate their functionality and data.
- Functionality is implemented without redundancy, and information is consistent.
- Functionality and data can be used everywhere within the domain, and they can be combined to support new business processes.
- New projects can build upon existing assets, and investments are secured.

One of the first services to be realized was *Customer Management*, that is, management of core customer data. This service was chosen because it is widely used within Deutsche Post. It offers about a dozen operations, including operations for inserting, searching for, or deleting customer data. Currently, there are around ten service consumers of the customer-management service with a resulting workload of approximately 0.5 million calls per month.

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 14.2. Implementation

As mentioned previously, the main reason for implementing the SOA at Deutsche Post was the realization that the IT landscape existing at that point in time was too expensive and complex to maintain and extend. Interestingly, this was not due to the existence of mainframes and host applications, as is often the case in large corporations. No such systems were in use at Division Mail. Complexity simply stemmed from the interfaces and the high degree of intertwining between the existing point-to-point interfaces and applications.

<sup>177</sup> This is further evidence for the fact that the quality of a software architecture is largely independent of technology. Both brand new developments comprising Java, C++, and decentralized servers and traditional environments based on COBOL, C, and centralized mainframes can equally serve as a base for an efficient architecture.

As this complexity led to the failure of projects at Deutsche Post, it prompted a change in the infrastructure. No real alternative to an SOA was investigated in detail, as no other option seemed sufficient. At the beginning of the SOA introduction, costs for the initial business projects and for the Service Backbone development were estimated. However, no detailed business case was compiled, as it seemed obvious that changing the infrastructure into an SOA was inevitable.

Although there was no resistance on the conceptual level against the SOA approach, some problems arose as soon as implementation started. The next section describes the processes and structures used by Deutsche Post to overcome these challenges.

#### 14.2.1. PROCESSES AND STRUCTURES

Deutsche Post differentiates two major types of components in its SOA. There are service providers, which offer services, and service consumers, which use those services. Obviously, a piece of code can act as a service consumer with respect to some services and as a service provider with respect to other services (see <u>Chapter 5</u> for the discussion of basic and intermediary services).

Deutsche Post defined a clear process for achieving a service design that was accepted across the borders of a single department (see [<u>HBa04</u>]). When implementing a new service provider, the following steps must be executed:

- Identify all potential service consumers.
- Specify business functionality of the new service according to the requirements of all potential service consumers.
- Match business requirements to Business Domain Model and implemented services.
- According to the findings, define and start an implementation project for this new service provider.
- Create a service description, service-level agreement, and an XML Schema for publishing the service to potential service consumer.
- Connect service provider to Service Backbone by deploying the SBB interface locally.
- Register service provider in SBB user directory.

In order to connect a new service consumer to the SBB, the following two steps must be processed:

- Insert service consumer information in SBB user directory for authentication and service authorization.
- Connect service consumer to SBB by deploying the SBB interface locally.

As we mentioned earlier, when this process of service development was introduced at Deutsche Post, some problems emerged that were mostly related to division of labor and sharing of responsibilities. For example, when a project initiated the development of a service, all potential consumers of the service had to be consulted. On one hand, it was not easy to get future consumers interested, and on the other, it meant that actors "not paving" for the service development gained influence. In general, business units were

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

## 14.3. Technology

The service concept of Deutsche Post's SBB is similar to the Web service concept. Web technologies such as SOAP and HTTP are combined with reliable messaging through an underlying MOM-based on JMS (Java Messaging Service). Furthermore, the SBB supports important features such as version management, safety, and dynamic binding and thus offers enriched functionality compared to standard Web services. For the future, Deutsche Post plans to publish SBB services as Web services based on WS-I Basic Profile 1.0.

WS-I Basic Profile 1.0 is a profile developed by WS-I (Web Services Interoperability) to ensure interoperability between Web services platforms from different vendors (see http://www.wsi.org).

#### 14.3.1. ARCHITECTURE

The SBB is built of three key components, which we will describe in detail (see [HS03]):

- Local SBB interface
- Central infrastructure components
- Technical service participants

Local SBB interfaces enabling the connection are implemented in each service participant. There are two kinds of local SBB interfaces: Service providers use a Service Programming Interface (SPI), whereas service consumers are connected by an Application Programming Interface (API). When a consumer calls a service, it uses the API, sending a message that contains the input parameters. This message (XML) is sent to the provider's SPI by SBB, and the requested operation is started.

For the processing of a service call, two central infrastructure components are involved the Service Registry (currently based on LDAP, UDDI in development) and the message queue (based on JMS). To ensure high availability, Deutsche Post replicates and clusters these infrastructure components. Figure 14-4 shows a service call using SBB functionality.

This call information mainly consists of an XML document containing attributes of the business objects that are to be created, manipulated, or modified. Additional parameters can be used to control synchronous or asynchronous behavior or other aspects, such as the number of results to be returned when calling a search service.

The Service Registry is the central source for all information needed for the communication between service participants. It should be noted that all interfaces always access the Service Backbone, regardless of their interaction style, which can be synchronous, asynchronous, publish/subscribe, or request/reply. Which interaction type is used depends on the business requirements. When calling the interface, the interaction type is passed through a parameter.

The call to the Service Backbone is realized as a Java call, where the main argument containing the business object information is represented as an XML document. The structure of the XML documents is described through service-specific XML Schemas. Internally, SOAP is used on top of Java, and there are also wrappers and adapters for C++ and non-XML arguments.<sup>10</sup> Actually, before the Service Backbone initiative started, the IT Strategy of Deutsche Post was based on C++. The need to easily integrate the associated software assets using the aforementioned wrappers and adapters is a major requirement of the new SOA.

In For the Java wrappers of C++-Code the product Junc++ion from codemesh is used, which supports various C++ dialects, including GNU and .NET.

The Service Backbone itself is built in a loosely coupled fashion, relies on standards, and avoids proprietary features. Although for example MQ Series is used for message handling, no proprietary features are used. Instead, connection to MQ Series is realized by using the JMS interface. A similar approach is used with other components in order to ensure that products can be easily interchanged and no vendor-dependency is created by using non-standard features.

Deutsche Post currently uses various *technical service participants* including *Transformation, Service Registry Administration, Data Integration,* and *Single Sign On. Privilege Management* and *Service Registry* are candidates planned for further releases

#### Team LiB



#### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

# 14.4. Lessons Learned, Benefits, and Perspectives

After more than two years of successfully running its SOA and the associated service infrastructure SBB, Deutsche Post has gained much useful experience, which is summarized in the following practical advices [HBa04]:

- If you want to reduce IT complexity through integration, start with the business (logical) view and use an architectural approach.
- Focus on services (multiple use of data and functionality), not on data interchange (point-to-point communication).
- Don't neglect the integration infrastructure layerthis is much more than just "data transport."
- The technical integration infrastructure is characterized by long-term stability, so stay vendor-independent and stick to open standards.
- The success of a whole integration project mainly depends on the acceptance of all business-driven service participants. So, don't let the IT professionals drive the effort alone!

Deutsche Post managed to reduce time to market significantly. The realization of new services only takes a few days due to the integrated infrastructure. Using the SBB helps keep implementation projects manageable and lean. These projects can focus on service implementations and do not have to worry about data connectivity. Moreover, the SBB can be maintained and updated without having to modify the services themselves.

As we mentioned, there were also some challenges to overcome when introducing the SOA. In particular, the effort initially needed to convince service providers to take reusability into account in their development process was higher than expected. A mixture of persuasion, subsidies, and coercion was necessary to achieve the desired compliance with the SOA standards.

A particularly visible success story was the adoption of the service "*Customer Management*" in the call center application, which today provides additional functionality such as registered mail. Before this adoption, all call center contacts were manually recorded by the agents, which led to typos and data duplication. Now call center agents have direct access to customer data by simply typing in the customer identification number.

The SOA and the Service Backbone have proved to be successful at Deutsche Post and are constantly extended by involving current and potential users in the overall process.

According to [<u>HBr03</u>], the following extensions are planned for the next major release (SBB Release 3.0) of the Service Backbone:

- Instrumentation and service management
- Service call routing with intermediaries
- "Users and Rights" as a service participant
- Pluggable core architecture
- Support of "Process Integration" (phase 1) as a service participant

After the successful introduction of the SOA at Division Mail, a rollout at DHL is now under way. In doing so, Deutsche Post will reuse the basic methodology and the Service Backbone developed already. A special team will be set up at DHL that will be responsible for integration and consolidation. This team will be supported by the IT strategy team at Division Mail. The decision to extend the SOA to DHL is also a sign of the positive perception of the SOA at the level of Deutsche Post's top management.

### References

[HBa04] Herr, Michael and Uwe Bath . *SBB Motivation Paper: The business-oriented background of Service Backbone*. <u>http://www.servicebackbone.org/</u>, January 2004.

#### Team LiB



# **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

# **Chapter 15. Winterthur Case Study**

In this chapter, we will describe the Service-Oriented Architecture implemented by the Winterthur Group, a leading Swiss insurance company with its head office in Winterthur. As an international company, the Group provides a broad range of property and liability insurance products in addition to insurance solutions in life and pensions that are tailored to the individual needs of private and corporate clients. The Winterthur Group has approximately 20,000 employees worldwide, achieved a premium volume of 33.5 billion Swiss francs in 2003, and reported assets under management of 138.7 billion Swiss francs as of December 31, 2003.

In 1998, Winterthur's Market Unit Switzerland developed a concept for an Application Service Platform. Since then, this application and integration platform called "e-Platform" has been implemented and used as the technological basis for the realization of a Service-Oriented Architecture. Its main purpose is to provide a common set of standards, guidelines, processes, frameworks, and integrated products in the form of a single package suite to access functionality available on the mainframe through standardized service interfaces. This functionality is then used to provide customer data access, claims notifications, financial reports, life insurance quotations, analysis and management of company risks, and information systems for insurance brokers.

The main focus of Winterthur's SOA is to provide reusable coarse-grained and technology-independent services for the application frontends in order to enable the access of backend functionality on the mainframe. This matches the purpose of an SOA, which is to decouple the existing system components by applying the principles of modularity and encapsulation.

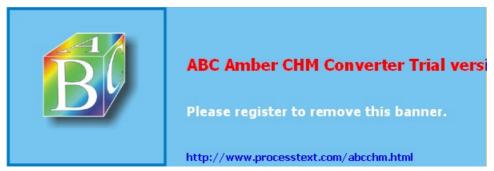
The main business driver for the SOA introduction was Winterthur's plan to offer their customers, partners, and employees new channels, in particular access using the Internet/Intranet, which required a tighter integration of existing functionality. The monolithic mainframe system provided a major obstacle to those plans, and therefore they decided to use an SOA to start it. They hoped that the SOA, which was technologically based on CORBA, would significantly reduce the overall complexity of the system and help to lower soaring maintenance costs. It was the desire to reuse as much of the implemented services as possible.

In the meantime, the platform has become a suite of integrated software infrastructure technologies, consisting of an integration framework, a portal framework, a security framework, and enterprise application servers. Today, it is not only used in Switzerland but also abroad in other Market Units of Winterthur.

The case study presented in this chapter will show in some detail how the SOA has been implemented at Winterthur, both at the organizational and technical levels. <u>Section 15.1</u> describes the general scope of Winterthur's architecture. <u>Section 15.2</u> discusses the organizational structures and processes used to implement the SOA. A more technological perspective is presented in <u>Section 15.3</u>, and finally, <u>Section 15.4</u> describes the lessons learned, benefits achieved, and future enhancements for the company.

#### Team LiB

▲ PREVIOUS NEXT ▶



Team LiB

# 15.1. Project Scope

The scope of Winterthur's SOA introduction is mainly defined by the requirements of innovative business-driven projects and the need to reuse existing mainframe applications.

### **15.1.1. BUSINESS IMPACT**

The first pilot project to be implemented within the SOA was wincoLink, an interactive customer service of Winterthur Leben, the life insurance branch of Winterthur. It provided corporate customers with online access to all contracts and contained direct information about, for example, vested benefits. It also supported changes of contract information, such as the inclusion of new employees in a corporate contract or a change to address data.

wincoLink was chosen as the pilot project because it not only was restricted to the passive browsing of static content but also involved user interaction. This provided Winterthur with the prestigious benefit of offering the first interactive life insurance Internet application. In addition, wincoLink promised significant cost-saving potential because it reduced Winterthur's customer support by enabling customers to access and change contract information directly, avoiding the need to go through Winterthur staff. In addition, wincoLink increased customer satisfaction because the online content available for inspection was always up-to-date.

Finally, wincoLink offered the advantage of a restricted user group, namely corporate customers, which could be enlarged step by step. It was thus possible to increase the support organization and the necessary processes incrementally, while extending the user group in a parallel manner. In fact, the wincoLink project turned out to be ideal for collecting experiences associated with the SOA without any major risks.

# **15.1.2. TECHNOLOGY IMPACT**

The focus of Winterthur's SOA is on the integration of existing host applications. One of the major incentives for the new architectural approach was the soaring cost for maintaining monolithic applications on the mainframe computer. In addition, Winterthur wanted to add new "sales" management channels to their IT infrastructure, in particular through the Internet and Intranet in order to make their applications and solutions widely available.

In Winterthur's SOA, a service is defined as a set of operations granting access to a simplified business context or to enterprise information in the form of core business entities. Winterthur distinguishes three types of services (the terminology used for this distinction is Winterthur's):

**Domain-specific business services.**<sup>III</sup> These services belong to a defined domain, using the domain-specific model to manage enterprise information. The focus is on reusing functionality related to core business entities. These services are implemented within the domain service layer that in return provides core business functions grouped by domains such as partner, product, contract, or claims. This function is subsequently reused across several applications, protecting the enterprise data by ensuring that business rules are correctly applied.

 $\ensuremath{^{\scriptsize (1)}}$  According to the terminology used in this book, these would be basic services.

**Services implementing business processes.**<sup>21</sup> These services orchestrate domain specific processes from different domains in order to provide functionalities and composite information for a single business activity. Business activities are the defined atomic steps within a business process. The focus is on providing a functional, simplified business process. Reuse, however, is not the main issue at this layer. Instead, these services are implemented within the application layer and are responsible for providing the business-process context to the domain service layer. In other words, this layer acts as a facade to combine and extend services to implement the business functionality described by use cases. This layer is accessed by the presentation layer that enables the user to interact with the system.

According to the terminology used in this book, these would be intermediary services, mostly facades.

**Technical services.** Technical services provide functionalities related to security or system

#### Team LiB



# **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

# 15.2. Implementation

This section deals with the processes and structures that Winterthur established to guarantee the success of the SOA, the repositories constructed to make information on services available, and the project management techniques employed.

Selling SOA within the Winterthur was difficult, especially explaining its benefits and its cost. It took also a relatively long time to make its specific concepts understood and to develop an understanding of the systems' implications and the necessary process adjustments.

Two reasons led to the initial support for the SOA. First, the problems resulting from the monolithic structure of the mainframe applications were blatantly obvious, in particular regarding maintenance costs and lack of flexibility. Second, architects, and analysts advertised for the SOA at all major IT events.

In particular, the local CIO strongly supported the SOA. Building on the previously available e-Platform infrastructure, a project team accomplished the necessary standards, guidelines, training modules, processes, and organization to define and implement the SOA. The project team was staffed with a resort architect, an e-Platform architect, e-Platform engineers, members of the software development support group and data management, host application developers, and an external consultant.

### **15.2.1. PROCESSES AND STRUCTURES**

Figure 15-4 shows the development process based on the Rational Unified Process proposed by the e-Platform. It distinguishes between a business view and a technological view on one hand and the three standard development disciplines (Requirements, Analysis & Design, and Implementation) on the other.

### Figure 15-4. e-Platform's analysis and design process.

The business view focuses on functional requirements and develops use case models and a component landscape. In doing so, it explicitly aims at designing *reusable* business components that provide their functions through services. The technological view deals with the non-functional requirements and concentrates on the reference architecture based on e-Platform blueprints. The application architecture is formed by the integration of the business view and the technological view.

Figure 15-5 shows the key aspects of Winterthur's design process in more detail. In order to capture the requirements, use case models, user interfaces, and conceptual business models are developed. These roughly corresponded to the three tiers of the architecturethe presentation layer, the application layer, and the domain service layer. The models are used as a basis for the realization of the use case and the service design, and they consider both static and dynamic aspects of the system. Whereas the static service design focuses on the service interfaces and data elements, the dynamic service design addressed workflow issues, that is, how service operations are to be combined in order to obtain the business activity-oriented services identified in the design of the use-case realization.

# Figure 15-5. Details of Winterthur's analysis and design process (focused on the business view).

[View full size image]

In order to ensure that the general design process is actually applied to specific services, a dedicated team called Application Services was established within the Winterthur Market Unit Switzerland. One of its tasks is to advise the application project teams on how to leverage the Service-Oriented Architecture in the best possible way. To do so, members of the group support the business developers when new services were designed. The group also offers training and instruction courses on its Service-Oriented Architecture, service-oriented design principles, and repository use. It is also responsible for QA on service definition

#### Team LiB



# **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

# 15.3. Technology

This section provides more detail about the technologies used to implement Winterthur's SOA and the e-Platform. Winterthur's host applications had been mostly developed in PL/1 and COBOL, and most program maintenance still requires PL/1 and COBOL (IMS and CICS on z/OS).

## **15.3.1. ARCHITECTURE**

Figure 15-7 shows the different architectural issues to be dealt with in the technical part of an application specification.

### Figure 15-7. Technical part of application specification.

[View full size image]

Figure 15-8 provides a detailed overview of the e-Platform's internal structure. It consists of HTML, Web services, Java clients in both Internet and intranet, secure proxies screening high-level communication protocols at the entry of the Intranet, Web and application servers, and enterprise information systems.

### Figure 15-8. Internal structure of Winterthur's e-Platform.

[View full size image]

A key concept underlying Winterthur's SOA and its e-Platform are *blueprints*. These blueprints are reusable reference architectures that propose standards concerning how business components can be distributed and integrated. They specify technical aspects of platform-specific environments, components, and protocols for various distribution patterns.

Figure 15-9 contains two sample blueprints, one employing a remote communication between an EJB and a CORBA service, and the other using a local communication between an EJB and a domain service implemented in Java.

### Figure 15-9. Sample blueprints for Winterthur's e-Platform.

The e-Platform contains blueprints for Java Clients, Servlets, EJBs, CORBA, Message-oriented middleware, Database connectivity and File transfer.

# **15.3.2. REPOSITORY, SERVICE INTERFACES, AND CONTRACTS**

The repository contains two main types of information: the descriptions of the enterprise data elements at the level of attributes and the specification of services (see Figure 15-10). The data element descriptions are reused for the database definition and for the definition of the parameters within a service. All data elements must be approved by a central unitthe data management team.

# Figure 15-10. Service definitions are based on enterprise data element definitions.

The application service team uses these data elements to define, in close cooperation with the respective service owners, the detailed service specifications. Data elements and service information are accessible for all developers using browsers within the Intranet.

So far, the focus of Winterthur's SOA development has been on synchronous services offering a request-reply function. These services provide IDL interfaces and have been implemented as CORBA services. It should be noted, however, that the contracts are

#### Team LiB



# **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

# 15.4. Lessons Learned, Benefits, and Perspectives

The introduction of the SOA has already delivered substantial benefits. The development of new applications has been significantly simplified due to the flexible nature of the implemented services and the resulting reusability.

What is particularly noteworthy is the fact that Winterthur has achieved these benefits by using very simple and basic SOA concepts: service orientation, explicit contracts, reused policies, and a descriptive but concise repository. Winterthur did not employ any advanced SOA concepts, such as service composition or distributed transactions.

One of the major success factors was the efficient process established at Winterthur to ensure reusability of developed services and e-Platform blueprints. However, it also became clear that designing services with focus on reusability generates a considerable overhead. It transpired that only one in three or four services was actually used by more than one application. Additional new application frontends, however, are enhancing the reuse rate. A further lesson was learned: design focused solely on reuse can lead to overly fine-grained services (e.g., to have an overview of a customer or a contract, you might have to call many fine-grained services to get all information related to an overview). Performance will be less than optimal if the service is accessed remotely, which leads to performance-optimized remote services that internally called fine-grained services accessed by local interfaces. The same fine-grained services can be easily encapsulated by a CORBA interface and called by a remote client. Further optimization was found in the so-called "multiple" services. Rather than retrieve a single contract or person through a single key, a whole list of business entities can be obtained with a single remote call using a list of keys.

Also due to performance issues related to remote communication, both domain layer services using CORBA and in some cases application layer services<sup>10</sup> were implemented on the host.

Process-centric services according to the terminology of this book.

One way of minimizing the overhead caused by reusability is to explicitly distinguish between public and private services. Only the former are designed with reusability considerations in mind, whereas the latter are to be used solely by the application for which they were originally developed.

Apart from these qualifications, however, the reuse of implemented services was rather successful. All applications using host data are migrating to use them through the newly developed services. The SOA has therefore become the cornerstone of Winterthur's IT integration.

Another major benefit is the widespread use of the repository. The information available in the repository turned out to be an excellent documentation of the already implemented functionality. In contrast to traditional documentation that quickly becomes complex and voluminous, the information contained in the repository is very concise. This is mainly due to the fact that the information to be published in the repository is restricted to essential facts required to adequately use the service in applications. On the other hand, the simple fact that the repository imposes a standardized format also contributes to its usability and offers an advantage over traditional documentation, which is usually crudely structured.

The development of Winterthur's SOA and its underlying e-Platform still continues. The main direction of enhancements concerns the removal of platform limitations, in particular regarding the SOA support of message-type communication, EJBs, and Web services.

Whereas emphasis has been on host applications in the beginning, focus now shifts to the application layer and non-host applications. Because the application layer is largely based on EJBs, the main task is to extend the SOA standards, guidelines, and processes that are currently based on synchronous CORBA to encompass EJBs, asynchronous messages, and Web services.

Another area of extension concerns workflows. To date, workflows are not explicitly modeled and supported in the e-Platform. They are only contained in the dynamic models of use case realizations developed in the design phase. The integration of workflows to support specification, automatic execution, monitoring, and optimization of workflows is currently under investigation.

#### Team LiB



# **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

# **Chapter 16. Credit Suisse Case Study**

In this chapter, we will describe the introduction of a Service-Oriented Architecture at Credit Suisse. The Credit Suisse Group (CSG) is a leading global financial services company headquartered in Zurich. The business unit Credit Suisse Financial Services provides private clients and small- and medium-sized companies with private banking and financial advisory services, banking products, and pension and insurance solutions from Winterthur. The business unit Credit Suisse First Boston, an investment bank, serves global institutional, corporate, government, and individual clients in its role as a financial intermediary. Credit Suisse Group's registered shares (CSGN) are listed in Switzerland and in the form of American Depositary Shares (CSR) in New York. The Group employs around 60,000 staff worldwide. As of March 31, 2004, it reported assets under management of CHF 1,241.3 billion.

Given the magnitude of operations, why did CSG decide to introduce an SOA? At the end of the 1990s, the complexity of the Credit Suisse IT infrastructure reached a critical level. The CIO made the decision to introduce an integration architecture based on a Service-Oriented Architecture. After the successful introduction of an information bus providing synchronous communication, Credit Suisse added an event bus for asynchronous communication. Whereas the information bus connects host applications with application frontends, the event bus is used for backend to backend integration. Currently, a third type of integration bus operates using file transfer for communication.

Essentially, this creates an environment very similar to the one outlined in Chapter 9, Figure 9-2. The notable difference is that rather than one software bus, three different busses are used. However, unlike in Figure 9-3, only one software bus technology per communication model is used.

In general terms, the authors of this book are critical of the integration bus concept. Firstly, the requirements for such an integration bus are usually too diverse to be covered by a single, homogeneous framework. Secondly, innovation cycles for products and standards in this area tend to be very short.

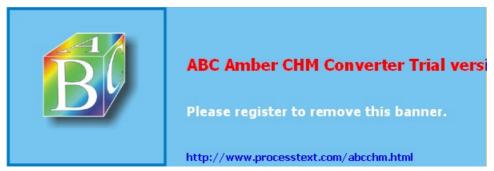
The approach taken by CSG, however, turned out to be very clever. They defined and implemented an integration bus according to their most pressing needs and obtained immediate benefits. They subsequently implemented a second bus that was based on similar principles but that satisfied slightly different technical requirements and therefore provided complementary benefits.

It is also noteworthy that this case study is very well complemented by various articles that provide in many respects an even more detailed discussion of the technology and the architecture (see [Ha03], [FMP99], [KM99]).

The case study presented in this chapter will show in detail how the SOA was implemented at Credit Suisse, both at organizational and technical levels. Section 16.1 describes the general scope of the Credit Suisse architecture, Section 16.2 discusses the organizational structures and processes used to implement the SOA, Section 16.3 presents a more technological perspective, and finally, Section 16.4 describes the lessons learned, the benefits achieved by Credit Suisse, and future perspectives.

#### Team LiB

♦ PREVIOUS NEXT ►



Team LiB

# 16.1. Project Scope

The central book entry system of Credit Suisse comprises approximately 5 million accounts with roughly 218 million account movements per year. The service architecture described in this chapter covers the banking business. Winterthur, who also belong to the Credit Suisse Group, have their own IT infrastructure.

<sup>[2]</sup> See Chapter 15 for the Winterthur case study.

The Credit Suisse IT infrastructure is typical of a large financial corporation and comprises around 600 applications, approximately 12 million lines of code (counting only core systems), and an application landscape based on heterogeneous platforms (IBM mainframe, Unix, Windows) grown over several decades. The roots of its IT systems reach back to the 1970s and terminal-based applications. At the end of the 1980s and the in the early 1990s, client/server applications were added, based on the then-innovative 4GL generators and object-oriented technologies using Smalltalk. With the rise of the Internet and intranets, multi-tier architectures were favored, and today, new applications are mostly built using Java. However, mainframe applications are still supported and updated, and the mainframe continues to be the preferred platform for transaction-oriented applications.

### **16.1.1. BUSINESS IMPACT**

The main driver for the SOA introduction at CSG was the fact that the former IT infrastructure could no longer support the required business functionality. Ad hoc solutions for integrating IT systems, such as after mergers and acquisitions, were not successful.

Credit Suisse was dissatisfied with point-to-point integrations. This had made development of new applications extremely complex and sometimes unfeasible. Although no business plan was compiled prior to introducing the SOA, the decision for the SOA was made directly by the CIO in connection with two other major projects at Credit Suisse: the reconstruction of the data centers, which was necessary due to a number of mergers and acquisitions, and the clearing up of the data warehouse.

In fact, Credit Suisse suffered from the classical middleware overload created by too many products and point-to-point connections. Scenarios like that ultimately create the famous "integration spaghetti."

The SOA introduction started with small pilot projects in 1997, but it was the intention from the very beginning to use it as a basis for the overall IT infrastructure of Credit Suisse, in particular for providing host access functionality.

From the business point of view, the SOA infrastructure should become the basis for

- Multi-channel banking
- Online trading
- Consolidation of the core-business application portfolio

As the foundation of its SOA Credit Suisse designed a business-critical infrastructure that was meant to provide

- Centralized administration and management
- 24-7 operations
- Support for several thousands concurrent users
- High throughput
- Sub-second response time

The applications built on top of the new infrastructure were supposed to provide access to customers over the Internet and to employees over the intranet. This included all types of clients. Finally, extra gateways were built to realize B2B integration with partners over the Internet.

### **16.1.2. TECHNOLOGY IMPACT**

#### Team LiB



# **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

# 16.2. Implementation

Although the CIO backed the SOA, uncertainty remained, mainly regarding technical issues. The main problem was its complexity and the overheads it created. Specifically, ensuring reusability was considered by many as a major factor in the increase of the design process cost. In addition, technical objections to the use of CORBA, which is the base of CSG's service bus, arose. Host developers believed it to be too resource-intensive, and Java developers argued that a pure J2EE approach would be more slender and thus preferable.

From the outset, Credit Suisse took this opposition seriously. The architecture was given a strong, central position within the IT department, and it was made clear that exceptions and deviations from the chosen approach would not be tolerated. Reuse was demanded, documented, and aggressively marketed, as was decoupling of systems.

A strict pursuit of the aims of the SOA and the application of rigorous processes helped to make the Credits Suisse SOA a success, and most opponents were eventually convinced by the benefits obtained by the SOA.

### **16.2.1. PROCESSES AND STRUCTURES**

Credit Suisse established two infrastructure teams dedicated to the establishment of the SOA-based integration architectureone responsible for engineering the required middleware, and the other supporting developers using the technology. These teams were responsible for several different though related tasks and were supported by integration architects from the central architecture department (see Figure 16-4).

# Figure 16-4. Different teams at Credit Suisse support the projects and maintain the SOA infrastructure.

First, the teams had to set up processes and structures accompanying the SOA introduction. In particular, this concerned stipulations for the service contracts and interfaces, in addition to the definition of a clear design and development process for services.

Second, the team had to educate and support developers from the different business units with respect to these concepts. The central challenge was to sell the concept of reusability, which at first glance, generated nothing but overheads from the perspective of developers and individual projects.

Finally, the teams were responsible for reviewing service definitions and thus acting as a kind of quality assurance. Again, reusability was the key concept here and was also the distinguishing factor compared to "traditional" project management. The team had to ensure that service development followed the established processes and fulfilled the requirements imposed by the integration architecture. In particular, it had to ensure that business units did not succeed in circumventing the SOA and get permission from management to make "exceptions."

One of the most important aspects of the SOA introduction was the establishment of a clearly defined process for service development. This process started with a communication between service consumers and service providers and resulted in a coarse-grained specification.

Based on these specifications, a decision was taken to either develop a new service or to use an already available service, which potentially had to be modified or extended to fit the new application. The architecture board reviewed this decision before the design and implementation of the service could commence.

The architecture board is composed of experienced service designers from the central architecture group and from development support.

Service development proceeded in a strict bottom-up mannerno prior planning of the service landscape took place. Instead, a service would be defined and implemented whenever a specific client application required the respective functionality (see Figure 16-5

Page 23

#### Team LiB



# **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

# 16.3. Technology

This section describes in more detail the technology used to construct the SOA at Credit Suisse. It comprises a sketch of the architecture used to implement the information and event buses, an overview of the repository structure, the contracts and the service interfaces, and a summary of how security, workflows, and management are handled.

### **16.3.1. ARCHITECTURE**

As we mentioned earlier, the integration architecture deployed at Credit Suisse combines three different integration paradigms.

Whereas the Credit Suisse Service Infrastructure provides synchronous communication and is used for providing frontend access to host applications, the Event Bus Infrastructure (EBI) uses asynchronous communication to integrate new non-host applications. Finally, the Bulk Integration Infrastructure uses file transfer for communication.

#### 16.3.1.1 Synchronous Integration with the CSIB

When introducing the Information Bus, CSG began with an initial structuring of the existing applications, which was achieved by partitioning the applications into approximately 20 domains (refer to Figure 16-1) where an application domain combines all data and applications belonging to a certain business area. Figure 16-7 shows how applications are encapsulated inside domains. Whereas coupling is tight within a domain, it is loose across domains where coupling uses the information bus.<sup>41</sup>

<sup>(4)</sup> The concept of a domain at CSG is largely similar to the notion of service in this book. The approach taken by CSG is somewhat similar to the approach laid out in <u>Section 10.2.1</u>, "Service Enablement," and depicted in <u>Figure 10-4</u>. However, CSG decided not to make an effort to refactor any logic within the domain or between domains, where communication was not through one of the service buses. Thus, the actual applications remain in fact tightly coupled, even if the service interfaces might not expose the coupling to the client. Over time, replacement and upgrades of the underlying application and changes in the inter-domain communication models might facilitate an adoption of decoupled infrastructure without any significant impact on the already existing service clients.

# Figure 16-7. Partitioning of applications into domains, which are loosely coupled.

The CSIB was first implemented using CORBA technology. Figure 16-8 provides a detailed overview of the initial architecture (which did not include the Event Bus) and the respective technologies used for the different layers.

### Figure 16-8. Initial implementation of the Credit Suisse Information Bus.

As an alternative to CORBA, DCE and DCOM were evaluated but discarded. The integration of CORBA and EJBs used to implement the application frontends was built by Credit Suisse. Due to strict abstraction from the underlying technology, CORBA could in principle be replaced by another technology, such as Web services. So far, experiences with CORBA have been mainly positive, and it is still used in implementing new services.

#### 16.3.1.2 Asynchronous Integration with the EBI

In 2000, when the Information Bus had been successfully introduced and had proven to be robust and scaleable, Credit Suisse decided to add a second integration platform. The basic idea was to address backend-to-backend application integration (within one domain or across different domains) with the same basic concepts that had proven successful when introducing the SOA and the Information Bus.

Credit Suisse calls its approach of adding an event bus to the Information Bus a generalization of the SOA toward a component-based architecture. They reserve the term "service" for the synchronous communication used in the Information Bus. However, the approach to SOA advocated in this book is much more generic and also comprises

#### Team LiB



# **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

# 16.4. Lessons Learned, Benefits, and Perspectives

The SOA implemented at Credit Suisse is now firmly established within the enterprise and is considered to be a major success. The main benefits experienced can be summarized as follows:

**Reuse of services.** The business services are used across applications. Although the average reuse factor is only 1.6 when taking into account all services, some business services are used by up to 12 applications. The low average factor is mainly due to the fact that many services are currently used by a single application. Because services are built as soon as there is a single user, it can take some time before a second user materializes. Reuse is driven by the centralized repository containing the service interfaces and detailed documentations.

**More efficient application development.** Due mainly to the reuse of services, application development has been accelerated considerably. Usually, when a new application is under development, 75 to 80 percent of the required services are already available in the repository. This improves time-to-market of new solutions dramatically and also offers significant cost savings for the development process.

**Increase of collaboration.** Another benefit consists of the increased collaboration between the business unit developers and the programmers implementing the core business applications. It was also observed that experienced PL/1 programmers, who had become demotivated over the years, participated actively in the development process.

However, these benefits were not achieved without hard work. For one thing, there was continuous uncertainty regarding the approach taken with the integration architecture. This included, for example, complaints that CORBA was too resource-intensive, too complex, and too slow. This objection was not broad-based, however, and the consistent support from top management overcame it.

There was also a critical stage when the Information Bus threatened to fall victim to its own success. As more users accessed the applications built on top of the CSIB, performance, reliability, and availability became indispensable. Again, having management backing and sufficient budget helped to overcome problems during this critical phase.

It also transpired that the decoupling, which had been achieved for internal integration, did not necessarily suffice for external integration, which posed even more demanding requirements on the degree of decoupling.

Finally, the strict bottom-up approach applied throughout the development of the SOA will probably be complemented by top-down considerations in the future. This included more systematic decisions concerning reuse and more specific targets for service developers. One idea is to reduce the overhead for the development of services that might never be reused. Another aspect is the identification of "missing" services, even if they are not immediately needed by an application.

Credit Suisse stresses four main SOA aspects that were crucial to its success:

- Interfaces
- Processes
- Management commitment
- Solid technology

Evidence for the success of the Credit Suisse SOA-based integration architecture is based on the fact that the concepts and methodologies initially developed for the synchronous information bus could be reused one-to-one when introducing the asynchronous event bus. Furthermore, the implementation of the Bulk Integration Infrastructure is also based on the same foundation. This demonstrates that both the concepts and the methodology actually produced the desired results and that they are independent from the underlying technology.

### References

#### Team LiB



# **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

# Chapter 17. Halifax Bank Of Scotland: IF.com

Halifax Bank of Scotland (HBoS) is a UK Financial Services provider with divisions in Retail Banking, Insurance & Investment, Business Banking, Corporate Banking, and Treasury. HBoS is the UK's largest mortgage and savings provider with a customer base of about 22 million. The annual results of 2003 reported £3.8 billion profit before tax and £408 billion assets under management. HBoS group was formed through the merger of Halifax and Bank of Scotland.

Intelligent Finance was launched as a division of Halifax plc with the aim of attracting new customers from outside Halifax and specifically to target the UK clearing banks. Intelligent Finance was launched as Project Greenfield in 2000, starting an entire new banking operation from scratch. Three years later, by the end of 2003, Intelligent Finance had 820,000 customer accounts, representing assets of £15.5 billion. In March 2004, Intelligent Finance announced that it had broken even in 2003the project had been a huge success.

In order to prevail in a highly competitive market, a unique product concept had to be devised, enabling customers to link a range of personal banking productsmortgages, credit cards, personal loans, savings, and current accounts any chosen combination with interest charged only on the difference between their debit and credit balances.

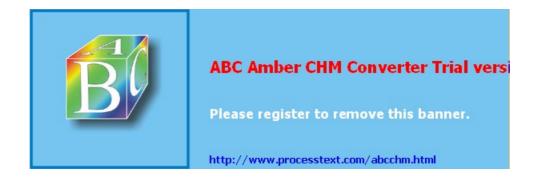
In order to enable Intelligent Finance to provide cost-effective products, it was decided to use only direct channels that is, not to rely on expensive branch offices. Because market research at the time showed that customers would prefer to do business with a bank that combined Internet banking with phone banking, it was decided to offer a solution that combined telephone and Web access.

At the heart of the Intelligent Finance system is a generic banking engine, which offers access to products and services for the different customer access channels. The Intelligent Finance system was probably one of the largest and most advanced SOA deployments in the Financial Services industry in Europe at the time. Because of time pressure under which the project was deliveredit took almost a year for the complete implementation of the bankit was decided early on in the project to take a full-blown SOA approach for its transactional aspects. The banking engine provides a suite of XML Web services, processing over 1,000,000 SOAP transactions a day. This chapter will take a closer look at the history of the project, the impact that the SOA approach had on the overall architecture, and the lessons learned in the project.

In Intelligent Finance adopted SOAP in the very early stages of the specification process. The SOAP version in use was upgraded several times throughout the project, in order to stay in line with the development of the SOAP specification. Service interfaces where initially defined in a proprietary, XML-based IDL, which was later updated to the emerging WSDL standard.

#### Team LiB

♦ PREVIOUS NEXT ►



Team LiB

# 17.1. Project Scope

Before delving deeper into the technical details of the Intelligent Finance project, we will look at the scope of the project from both a business and technology point of view.

### **17.1.1. BUSINESS IMPACT**

In 1999, Halifax was one of the most respected banks in the UK but was also overshadowed by the "big four"Barclays, Royal Bank of Scotland, Lloyds TSB, and HSBC (Halifax ranked fifth at the time). These four banks together controlled over 80% of the UK banking market in 1999. In order to attack the market share of the "big four," Halifax needed a distinct offering, which would differentiate Halifax in the market, as shown in Figure 17-1.

### Figure 17-1. Competition and customer demand were the main drivers for Halifax to move toward new innovative banking products and modern access channels.

At the same time, the UK banking market had seen something of an e-banking gold rush in 1999, with the Co-operative Bank's Smile, followed by Prudential's Egg, HSBC First Direct's 'Little Fella,' and Abbey National's Cahoot launched in quick succession.

As a result, Halifax was under huge pressure to deliver their new bank in an extremely tight timeframe. Halifax management at the time estimated that they would have about one year to execute their plan and create a product that was good enough to succeed in this highly competitive market.

#### 17.1.1.1 Greenfield Project

In order to meet these challenging timelines, Halifax decided to invest GPB 120 million to build the new bank. In October 1999, Halifax hired Jim Spowart, the former chief executive of Standard Life Bank, to head the new company, which was initially called *Greenfield*. Three months later, the new bank identity was rolled out with the brand *Intelligent Finance*.

The benefit of being given a blank sheet of paper to create what was internally dubbed the *bank of the future* was that there were no legaciesthe IF management team was free to reinvent the ways the bank should look and how it should interact with its customers.

But there were also some obvious challenges and disadvantages in the Greenfield approach: There were literally no existing structures or processes; everything had to be invented from scratch.

### 17.1.1.2 Offsetting

In order to differentiate itself in the market, Intelligent Finance adopted a new concept, called *offsetting*. CEO Jim Spowart and his team developed the concept of inter-linked accounts, which they called *jars*. These jars would allow customers to see how the money in their debit and credit balances measured up. They envisaged an offsetting function across all products. As a result, customers are only charged interest on the money they actually owe the bank. For example, if a customer had borrowings of £150,000 and £50,000 in savings and/or a current account with Intelligent Finance, interest would only be charged on the £100,000 outstanding loan, in return for no interest being charged on the savings or current account. Because no interest is earned on credit balances, the customer is not required to pay tax. Over the term of the loan, this can save thousands in interest charges and enable the customer to pay off the loan early.

### 17.1.1.3 The IF.com Success Story

As we mentioned earlier, since it fully launched in November 2000, Intelligent Finance has been a huge success. In November 2001, Intelligent Finance announced that it had a total of £8.9 billion in balances in hand and forecast to complete. Savings and current account balances amounted to £2 billion.

#### Team LiB



# **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

# 17.2. Implementation

In this section, we describe the key implementation details of the Intelligent Finance project, including the service implementation concept, service repository, and project management.

### 17.2.1. XML SERVICES

Because of the extremely tight schedule and the high integration requirements of the multi-channel architecture, it was decided early on in the project that existing EAI blueprints and middleware technology would not be suitable for this project, due to their long and complex implementation cycles.

XML had just emerged as a new and flexible toolkit that enabled high productivity and provided very good ad-hoc integration features. It was therefore decided to use XML as the *lingua franca* within the technical architecture. However, although the great flexibility of XML provided a huge benefit over more stringent integration technologies such as CORBA, this flexibility also represented a problem in many respects. Especially in an environment where requirements are changing on a daily basis, it is often tricky to strike a good balance between flexibility and strictness.

Approximately 250 different service request types exist in this system. A way was needed to leverage the XML technology to model the behavior of the Intelligent Finance banking engine, which was at the heart of the system architecture. WSDL and SOAP were new standards at the time, but the architecture team decided to adopt them to specify the interfaces of the banking engine anyway. The often-problematic experience with distributed object systems that the architecture team had made in many previous projects naturally led to the adoption of a Service-Oriented Architecture. Even if the term was not defined at the time, the underlying concepts were applied in the design of the bank.

## **17.2.2. SERVICE REPOSITORY**

Intelligent Finance uses the CCC Harvest source control system as the central service repository for all service definitions used by the project. All services are defined as XML Schema definitions and WSDL definitions.

The service repository and the service definitions in it are managed by a technical architect who holds the role of *XML Tsar* (this role is described in more detail in the following project management section). The XML Tsar works with the different development streams as well as the business side to develop and maintain the service definitions.

The content of the service repository is used by the IF.com build manager to generate type-safe APIs and stubs for a variety of different programming languages, including Java, C++, and VB (see Figure 17-5). These stubs enable client- and server-side programmers to write service components and access them in a transparent way. These stubs are managed in separate repositories together with the actual source code of the application. One can debate whether it makes sense to actually manage the generated code in a repository because one should be able to regenerate the stubs at a later point in time. However, there is a danger that the exact version of the compiler used for the particular build in question might not be available any more, and therefore there is a danger that one would not be able to reconstruct an older version of a build. For this reason, it was decided to include the generated code in the source code repository.

### Figure 17-5. IF.com service repository.

### **17.2.3. PROJECT MANAGEMENT**

The huge scale and extremely ambitious schedule of this project put significant pressure on project managers. Starting from scratch, the Intelligent Finance management team had to build the entire Intelligent Finance organization, including HR, sales, marketing, legal, IT development and operations staff, call center staff and management, and the banking back-office. In parallel, every piece of infrastructure had to be put in place from scratch, including offices for the development team. plus the acquisition of two new buildings for

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

4 PREVIOUS NEXT +

## 17.3. Technology

Having discussed the implementation aspects of the project, we now want to take a closer look at the actual technology employed. This discussion will cover the technical architecture, XML service definitions, and technical infrastructure.

## 17.3.1. ARCHITECTURE

The technical architecture of the IF.com system required the integration of a wide range of heterogeneous technologies. The banking engine in the Mid-Tier is implemented based on Java and BEA WebLogic. The Web Channel (Web server to render HTML pages) is based entirely on Microsoft. This is due to the fact that IF already had a security approval for this Microsoft-based architecture based on their first-generation online bank. Call center and IVR (Interactive Voice Recognition) are based on the Genesys CTI suite, using customized C and C++ on Unix. In the backend, a variety of mainframe, Unix, and NT systems had to be integrated. Figure 17-6 provides a high-level overview of the technical architecture of the system.

## Figure 17-6. Technical architecture of IF.COM.

# 17.3.2. SERVICE REPOSITORY, SERVICE INTERFACES, AND CONTRACTS

As we discussed previously, the IF.com service architecture is divided into two main layers: a number of basic services in the backend, and a central service in the middle, which is a mixture of an intermediary and a process-centric service. All services are "hard-wired" through configuration filesthat is, there is no explicit service registry. Given the nature of the project (all services are controlled by the same project), this approach makes sense. The following describes the service operations and contracts of the banking engine service and the basic services in the backend.

## 17.3.2.1 Basic Services

The basic services implemented by the Intelligent Finance system are based on a number of very different technologies, ranging from CORBA to DCOM to XML and MQ Series.

Interestingly, Halifax itself started to develop a Service-Oriented Architecture for its mainframe-based core banking system, which was also used in the Intelligent Finance project. The so-called *message switch* for the Halifax mainframe is based on XML and MQ Series. A technique similar to the one described in <u>Chapter 3</u> is used to simulate synchronous service operations by using message correlation to group matching requests and responses together.

## 17.3.2.2 Banking Engine Services

The IF.com banking engine is based on approximately 1,300 XML Schema definitions, 120 WSDL Web service interfaces, and 600 Web service operations. Halifax is now processing over 1,000,000 XML SOAP transactions a day. This makes Halifax Intelligent Finance one of the biggest and most successful Web services projects today.

The banking engine service is divided into a number of different namespaces, including Common, ContactCentre, Workflow, OpenAccount, PersonalAdvisors, QuickQuote, and Service Request. The OpenAccount namespace, for example, includes service interfaces such as AddressMgr, ApplicationMgr, BroadRequest, OfferEngine, CreditCardApplication, CurrentAccountApplication, and so forth. The OfferEngine includes, for example, XML Schema definitions such as DebtType, MortgageProductDetails , and MortgageOffer. The OfferEngine service interface provides operations such as renegotiateMortgageOffer(), getMortgageOfferAcceptance(), and so on.

In general, the granularity of service operations is closely tied to the granularity of screens used by the Web channel and call center channel.

The banking engine service runs on a standard 12FF application server, using session

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

## **17.4. Lessons Learned, Benefits, and Perspectives**

Probably the most important lesson learned in the project was that putting a Service-Oriented Architecture in place requires not only a sound technical design but also a project management initiative that supports the technical architecture on the project level. The *XML Tsardom* that was established in the early phase of the project was a key management tool for coordinating the development of a large set of technical interfaces that spanned 23 different work streams. In effect, the concept of the *XML Tsardom* as deployed by Intelligent Finance adopted many of the concepts that we have outlined in our discussion on SOA as a driver for project management in <u>Chapter 13</u>.

Another interesting observation is related to the evolution of the design of the central banking engine service: During the development phase of the project, priorities were very different than during the following maintenance and enhancement phase. The initially relatively tightly coupled architecture made a lot of sense during the development phase, providing the different frontend developers with an easy-to-use, ubiquitous interface. However, the same design became more problematic during the maintenance phase, which required a much more loose coupling. This led to the break-up of the initial service design into multiple independent services, which helped reduce dependencies and provide the maintenance and enhancement process with much higher agility.

Finally, it is interesting to observe that about 90% of the functionality that exists today was developed in the first nine months of 2000. The focus today is on maintenance, third-party integration, and increasing system agility. The original technical and functional design provides an excellent foundation for these activities. The key architecture decisionin particular the decision for the Service-Oriented Architectureare still valid today and have provided Intelligent Finance with one of the most advanced IT architectures in the banking world.

## URLs

www.if.com

http://www.if.com/aboutus/media/keymilestones.asp

http://www.if.com/aboutus/media/press2004.asp

http://www.vision.com/clients/client\_stories/if.html

http://www.actiontech.com/library/Documents/GetDocs.cfm?ID=INTELLEXEC

http://www.lynx-fs.com/cms\_view/news\_details\_all.asp?article\_ref=46

Team LiB

♦ PREVIOUS NEXT ►



### Team LiB

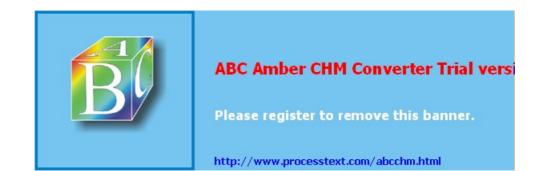
## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [M] [X]

### Team LiB

♦ PREVIOUS NEXT ▶

▲ PREVIOUS NEXT ►



#### Team LiB

▲ PREVIOUS NEXT ►

## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

#### 2PC 2nd distributed 2PC avoiding 2nd client controlled transactions 2nd 3rd implicit application level protocol 2nd 3rd server controlled transactions 2nd 3rd limitations discontinuous networks integration of legacy systems and packaged applications lack of support for long-lived transactions organizational challenges performance 2PC (Two-Phase Commit Protocol) \_Ref62470922 \_Ref62470929

#### Team LiB

♦ PREVIOUS NEXT ►



Team LiB

+ PREVIOUS NEXT +

## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

abstraction components functional decomposition access control lists (ACLs) access layers binding design rules ACID (atomicity, consistency, isolation, durability) ACID transactions limitations of integration of legacy systems and packaged applications lack of support for long-lived transactions organizational challenges performance ACLs (access control lists ADA ADA programming language adapters intermediary services adding service orientation to project management methodologies additional runtime features distribution techniques ADO DataSets ADO DiffGrams agility 2nd 3rd change requests TT application frontends SOAs 2nd application heterogeneity application landsca application level protocol distributed 2PC 2nd 3rd application servers 2nd applications multi-channel applications 2nd 3rd fundamental SOA 2nd process-enabled SOAs 2nd 3rd 4th 5th service facades 2nd 3rd architects perspective of SOAs SOA architects [See SOA architects] architectural roadmap fundamental SOA architecture 2nd 3rd BPM CSG asynchronous integration with EBI 2nd 3rd 4th Bulk Integration Infrastructure 2nd choreography 2nd contracts management 2nd repositorie security 2nd service interfaces synchronous integration with CSIB 2nd 3rd Deutsche Post case study 2nd 3rd 4th enterprise architecture versus standards 2nd 3rd Intelligent Finance 2nd 3rd multichannel architecture of enterprise software 2nd 3rd requirements of 2nd 3rd 4th architecture board service repository architecture boards architecture roadmap fundamental SOA 2nd 3rd 4th 5th 6th 7th 8th networked SOA 2nd 3rd 4th 5th 6th 7th process-enabled SOAs 2nd 3rd 4th 5th 6th 7th archtectural roadmap networked SOA archtiectural roadmap process-enabled SOAs asynchronous communication 2nd coupling asynchronous integration with EBI 2nd 3rd 4th atomicity auditing authenticating against SOA authentication 2nd 3rd 4th 5th 6th and middleware 2nd 3rd 4th 5th 6th creating SOAP 2nd 3rd 4th authorization 2nd 3rd 4th 5th 6th dynamic authorization static authorization automated test tools automicity, consistency, isolation, durability [See ACID]

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

+ PREVIOUS NEXT +

## Index

B2B 2nd 3rd 4th 5th 6th 7th

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [M] [X]

location transparency security infrastructures stateless semantics B2B (business-to-business) B2B integration Baan backers success of SOAs 2nd banking engine services Intelligent Finance 2nd basic layer basic services 2nd data-centric services 2nd 3rd 4th Intelligent Finance logic-centric services 2nd 3rd BDM Deutsche Post BDM (Business Domain Model) beans stateless session beans Berkeley r-tools suite BII (Bulk Integration Infrastructure) billing execution containers binding development-time binding runtime binding 2nd service binding binding design rules access layers Boehm, Barry Spiral Mode bonus systems BookAndBill 2nd Booking process booking process bottom-up code generation 2nd 3rd BPEL4WS (Business Process Execution Language for Web Services) BPM 2nd 3rd archecture of combining with SOA and MOA 2nd 3rd overview of modeling languages 2nd 3rd process integrity 2nd 3rd process-enabled SOAs 2nd core business logic versus process control logic 2nd 3rd 4th design implications 2nd The Third Wave (s/b ital) versus BPMS 2nd vision 2nd 3rd 4th 5th BPM (Business Process Management) 2nd BPML (Business Process Modeling Language) **BPMN** BPMN (Business Process Modeling Notation) BPMS versus BPM 2nd when to choose 2nd 3rd 4th BPMS (Business Process Management System) BPR (Business Process Reengineering) budgets success of SOAs 2nd Bulk Integration Infrastructure 2nd 3rd 5th 6th [See BII] business computing 2nd 3rd 4th SAP service-orientation Wal-Mart Business Domain Model (BDM) business exceptions business functionality complexity business impact CSG 2nd Deutsche Post 2nd 3rd 4th Intelligent Finance 2nd Greenfield Project 2nd IF.com success story offsetting Winterthur 2nd business infrastructure motivation for creating SOAs 2nd business level cost savings 2nd business logic SOA business logic Business Process Execution Language for Web Services (BPEL4WS) Business Process Management [See BPM] Business Process Management System [See BPMS] Business Process Modeling Language [See BPML]

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

+ PREVIOUS NEXT +

# Index

confirm itinerary conflicts of interest

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

callbacks callbacks and polling services callbacks and queues Carr, Nicolas G case studies Credit Suisse Group [See CSG] Deutsche Post [See Deutsche Post] HBoS [See case studies;HBoS] Winterthur [See Winterthur] central repositories centralized banking engine service CEO perspective of SOAs Champy, James change IT\_s ability to change change requests agility characteristics of enterprise software 2nd 3rd choosing BPMS 2nd 3rd 4th granularity for transactional steps 2nd 3rd 4th choreography CSG 2nd CICS availability 2nd scalability 2nd CICS (Customer Information Control System) CICS (Customer Information Control Systems) CICS log manager CICS Transaction Gateway (CTG) CIO conflicts of interests perspective of SOAs Class-Responsibility-Collaboration (CRC) classification 2nd 3rd client controlled transactions 2nd 3rd clients fat clients 2nd 3rd 4th clustering CM (configuration management) co-browsing COBOL (Common Business Oriented language) functional decomposition CODASYL (Conference on Data Systems Languages) code generation 2nd 3rd bottom-up approach 2nd 3rd top-down approach 2nd with MDA 2nd 3rd 4th combining SOA, MOA, and BPM 2nd 3rd transaction chains with compensating transactions Commodore PET Common Business Oriented Language [See functional decomposition;COBOL] Common Object Request Broker Architecture [See CORBA] communication asynchronous communication [See asynchronous communication] communication middleware [See communication; communication middleware] minimizing resources for communication simulated synchronous communication synchronous communication [See synchronous communication] communication middleware framework 2nd 3rd application servers 2nd Distributed Objects 2nd 3rd MOM 2nd 3rd 4th email RPCs 2nd 3rd transaction monitors 2nd 3rd communication modes compensating logic 2nd 3rd 4th compensating transactions combining with transaction chains compensation transactions 2nd complexity 2nd component programming components concurrency control optimisitc concurrency control 2nd 3rd 4th 5th example 2nd 3rd 4th 5th pessimistic concurrency control example 2nd 3rd Conference on Data Systems Languages (CODASYL) configuration management challenges for 2nd 3rd recommendations for the SOA integration team 2nd 3rd 4th 5th configurations logging 2nd 3rd runtime configurations

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

◀ PREVIOUS NEXT ▶

## Index

Dahl, Ole-Johan

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

```
data
  SOA
data access services
data integrity
   user-defined data integrity
   versus process integrity
data-centric services 2nd 3rd 4th
databases
DCE (Distributed Computing Environment) 2nd
debugging
decomposition
decoupling
   public enterprise services
decoupling from technology
decoupling of functionality and technology
   requirements of enterprise software architecture
design
BPM and process-enabled SOAs 2nd
Design in Action (DIA)
designing
   authentication
   for small devices 2nd 3rd 4th 5th 6th 7th 8th 9th 10th
Deutsche Post 2nd 3rd
   BDM
   implementing SOA 2nd
     processes and structures 2nd 3rd 4th 5th 6th
     project management 2nd
      service registry 2nd
   project scope 2nd
     business impact 2nd 3rd 4th
   technology impact 2nd
results of implementing SOA 2nd 3rd
   SBB
   SSB
  technology
architecture 2nd 3rd 4th
     contracts
     management
     repositories
     security
     service interfaces
Deutsche Post World net
development processes
  motivation for creating SOAs
development-time binding
DIA (Design in Action)
discontinuous networks
   limitations of 2PC
dispatching
   execution containers
distributed 2PC
   avoiding 2nd
     client controlled transactions 2nd 3rd
     implicit application level protocol 2nd 3rd
      server controlled transactions 2nd 3rd
distributed computing 2nd 3rd 4th 5th 6th
   SOAP
   XML
Distributed Computing Environment [See DCE]
Distributed Computing Environment (DCE)
distributed logging 2nd 3rd 4th 5th
Distributed Objects 2nd 3rd
distribution techniques
   heterogeneity 2nd
      additional runtime features
     communication modes
     products
distribution technology
   service-orientation
divide and conquer strategies 2nd
document-centric messages 2nd
documentation
   service documentation
domain
domain inconsistencies
domain-specific business services
drivers
   test drivers
durability
dX method
```

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

+ PREVIOUS NEXT +

## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] e-Platform 2nd EAI 2nd service enablement 2nd 3rd 4th 5th service repositorie service stability 2nd 3rd 4th SOAs upgrade ability 2nd 3rd 4th EAI (Enterprise Application Integration) 2nd EBI asynchronous integration 2nd 3rd 4th EBI (Event Bus Infrastructure EBI (Event Bus Infrastructure) eCommerce EDMs (Enterprise Data Models) EJB (Enterprise Java Beans) containers EJBs availability 2nd 3rd 4th scalability 2nd 3rd 4th email embedded messages [See payload semantics] encapsulation components Encina encryption 2nd 3rd enterprise application integration Enterprise Application Integration [See EAI] Enterprise Application Integration (EAI) Enterprise Data Models (EDMs) enterprise IT renovation roadmap 2nd 3rd 4th enterprise IT renovation roadmaps Enterprise Java Beans Enterprise Java Beans (EJB) enterprise laver Enterprise Resource Planning [See ERP] Enterprise Resource Planning (ERP) Enterprise Resources Planning [See ERP] **Enterprise Service Bus** enterprise software architecture versus standards 2nd 3rd architecture of 2nd 3rd requirements of 2nd 3rd 4th availability of characteristics of 2nd 3rd Enterprise Software Bus enterprise software systems lack of agility and inefficiency 2nd 3rd 4th enterprise-level software project management entity entity relationship models [See ER] entropy ER (entity relationship models) ERP (Enterprise Resource Planning) 2nd ERP (Enterprise Resources Planning error handling idempotent operations 2nd error reporting 2nd establishing project management methodologies 2nd 3rd 4th SOA-driven project management 2nd 3rd 4th Event Bus Infrastructure (EBI) 2nd evolution motivation for creating SOAs 2nd 3rd example scenarios travel itinerary management 2nd 3rd examples of optimistic concurrency control 2nd 3rd 4th 5th of pessimisstic concurrency control 2nd 3rd examples scenarios passenger check-in scenario 2nd 3rd exceptions out of stock exceptions execution containers cross-container integration 2nd 3rd logging message transformation security expansion stages fundamental SOA 2nd 3rd 4th networked SOA 2nd 3rd 4th 5th 6th 7th process-enabled SOAs 2nd 3rd 4th 5th 6th 7th exposing transaction logic to service clients Extreme Programming (s/b ital)

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

### Team LiB

◀ PREVIOUS NEXT ►

## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]



#### Team LiB

#### ♦ PREVIOUS NEXT ►



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

▲ PREVIOUS NEXT ►

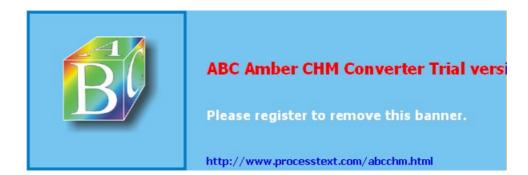
## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

geographic information systems (GIS) GIS (geographic information systems) goals of SOA granularity choosing for transactional steps 2nd 3rd 4th software artifacts Greenfield Project Intelligent Finance 2nd

Team LiB

▲ PREVIOUS NEXT ▶



#### Team LiB

▲ PREVIOUS NEXT ►

## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

Halifax Bank of Scotland [See case studies;HBos] Hammer, Michael hardware buses [See also software buses] HBoS (Halifax Bank of Scotland) Intelligent Finance [See Intelligent Finance] heterogeneity and security 2nd 3rd 4th 5th distribution techniques 2nd additional runtime features communication modes products

horizontal slicing versus vertical slicing 2nd hub-and-spoke

#### Team LiB

#### ♦ PREVIOUS NEXT ►



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

+ PREVIOUS NEXT +

## Index

intermediary services

adapters facades 2nd 3rd 4th

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] IAD (Iterative Application Development) IBM MQSeries 2nd IBM PC IBM WebSphere MQ ideal worlds SOA specifics 2nd 3rd 4th structures and processes 2nd 3rd 4th 5th 6th idempotent idempotent update operations 2nd 3rd error handling 2nd sequence numbers 2nd IDL (Interface Definition Language) lf.com Intelligent Finance IIS (Internet Information Server) implementation SOA implementing SOA at CSG 2nd processes and structures 2nd 3rd 4th project management 2nd 3rd repositories 2nd SOA at Deutsche Post 2nd processes and structures 2nd 3rd 4th 5th 6th project management 2nd service registry 2nd SOA at Intelligent Finance project management 2nd 3rd 4th repositories 2nd XML 2nd SOA at Winterthur 2nd processes and structures 2nd 3rd project management 2nd repositories 2nd IMS (Information Management System) inconsistencies domain inconsistencies process inconsistencies independence from technology 2nd 3rd Information Management System [See IMS] integration 2nd asynchronous integration with EBI 2nd 3rd 4th complexity synchronous integration with CSIB 2nd 3rd integration of legacy systems and packaged applications limitations of 2PC limitations of ACID transactions integration of purchased software Integration Spaghetti integrity business exceptions message integrity 2nd process integrity [See process integrity] special cases technical failures versus business exceptions 2nd Intelligent Finance architecture banking engine services 2nd basic services implementing SOA project management 2nd 3rd 4th repositories 2nd XML 2nd lessons learned from implementing SOA 2nd project schedule 2nd project scope business impact 2nd 3rd 4th 5th 6th technology impact 2nd 3rd 4th 5th 6th service layers 2nd 3rd technology architecture 2nd contracts repositories service interfaces 2nd interaction diagram showing check-in process for a Web application Interface Definition Language [See languages;IDL] interface semantics 2nd coupling versus payload semantics 2nd 3rd document-centric messages 2nd interfaces SOA intermediary layer intermediary service BookAndBill 2nd

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

#### ▲ PREVIOUS NEXT ►

## Index

 $[\underline{\mathsf{SYMBOL}}][A][B][\underline{\mathsf{C}}][\underline{\mathsf{D}}][\underline{\mathsf{F}}][\underline{\mathsf{F}}][\underline{\mathsf{G}}][\underline{\mathsf{H}}][\underline{\mathsf{I}}][\underline{\mathsf{J}}][\underline{\mathsf{K}}][\underline{\mathsf{L}}][\underline{\mathsf{M}}][\underline{\mathsf{M}}][\underline{\mathsf{O}}][\underline{\mathsf{P}}][\underline{\mathsf{Q}}][\underline{\mathsf{R}}][\underline{\mathsf{S}}][\underline{\mathsf{T}}][\underline{\mathsf{U}}][\underline{\mathsf{V}}][\underline{\mathsf{M}}][\underline{\mathsf{X}}]$ 

J.D. Edwards JZEE 2nd software buses J2ME SOAP JAAS (Java Authorization and Authentication Framework) jars Intelligent Finance Java Authorization and Authentication Framework (JAAS) Java Connector Architecture (JCA) JCA (Java Connector Architecture) Just-in-Time production

#### Team LiB

#### ♦ PREVIOUS NEXT ►



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

▲ PREVIOUS NEXT ►

## Index

 $[\underline{\mathsf{SYMBOL}}][\underline{A}][\underline{B}][\underline{C}][\underline{D}][\underline{E}][\underline{F}][\underline{G}][\underline{H}][\underline{I}][\underline{J}][\underline{K}][\underline{L}][\underline{M}][\underline{N}][\underline{O}][\underline{P}][\underline{O}][\underline{R}][\underline{S}][\underline{T}][\underline{U}][\underline{V}][\underline{W}][\underline{X}]$ 

key performance indicator (KPI) KPI (key performance indicator)

Team LiB

◀ PREVIOUS NEXT ►



#### Team LiB

◀ PREVIOUS NEXT ►

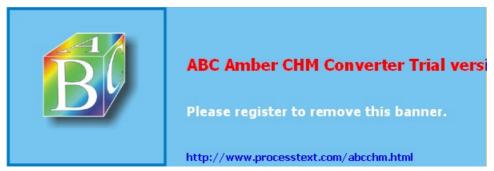
## Index

 $[\underline{\mathsf{SYMBOL}}][\underline{\mathsf{A}}][\underline{\mathsf{B}}][\underline{\mathsf{C}}][\underline{\mathsf{D}}][\underline{\mathsf{E}}][\underline{\mathsf{F}}][\underline{\mathsf{G}}][\underline{\mathsf{H}}][\underline{\mathsf{I}}][\underline{\mathsf{I}}][\underline{\mathsf{K}}][\underline{\mathsf{L}}][\underline{\mathsf{M}}][\underline{\mathsf{N}}][\underline{\mathsf{O}}][\underline{\mathsf{P}}][\underline{\mathsf{O}}][\underline{\mathsf{R}}][\underline{\mathsf{S}}][\underline{\mathsf{T}}][\underline{\mathsf{U}}][\underline{\mathsf{V}}][\underline{\mathsf{V}}][\underline{\mathsf{M}}][\underline{\mathsf{X}}]$ 

lack of support for long-lived transactions limitations of 2PC limitations of ACID transactions languages COBOL CORBA IDL modeling langues BPM 2nd 3rd MODULA Pascal SIMULA layers 2nd 3rd 4th 5th learning legacy software leveraging SOA to decompose complex systems thin thread model 2nd 3rd vertical versus horizontal slicing 2nd SOA to drive development iterations 2nd divide and conquer strategies 2nd managing parallel iterations 2nd 3rd limitations of 2PC discontinuous networks integration of legacy systems and packaged applications lack of support for long-lived transactions organizational challenges perfoormance of ACID transactions integration of legacy systems and packaged applications lack of support for long-lived transactions organizational challenges performance load balancers off-the-shelf load balancers interoperability local logging location transparency B2B Log Services log traces logging configurations 2nd 3rd distributed logging 2nd 3rd 4th 5th execution containers frameworks 2nd 3rd local logging transaction boundaries 2nd 3rd transaction logs transaction monitors logging and tracing process integrity 2nd 3rd logic compensating logic 2nd 3rd 4th core business logic process control logic process logic [See process logic] logic-centric services 2nd 3rd logical integration logs consolidated logs loose coupling 2nd 3rd 4th 5th 6th 7th

Team LiB

♦ PREVIOUS NEXT ►



Team LiB

+ PREVIOUS NEXT +

## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [**M**] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

maintainability requirements of enterprise software architecture maintenance complexity Manage Evolution management CSG 2nd Deutsche Post case study Winterthur 2nd 3rd managing [See also organizing] parallel iterationis 2nd 3rd service repository Manhattan Project Manifesto for Agile Software Development (s/b ital) Manugistics Market Units of Winterthur Martin, James RAD Martin, Robert dX method MDA code generation 2nd 3rd 4th MDA (Model Driven Architecture) 2nd message integrity 2nd message queuing systems Message Queuing systems message transformation execution containers Message-Oriented Middleware [See MOM] messages document-centric messages 2nd Meta buses meta buses creating Meta-Object Facility [See MOF] Microsoft ASP (Active Server Pages) middleware authentication 2nd 3rd 4th 5th 6th middleware heterogeneity 2nd minimizing resources for communication on small devices mitigating risk motivation for creating SOAs 2nd 3rd 4th 5th MMS (multimedia message service) MOA combining with SOA and BPM 2nd 3rd Model Driven Architecture [See MDA] modeling languages BPM 2nd 3rd MODULA modularization and component programming MOF (Meta-Object Facility) MOM 2nd 3rd 4th emai synchronous communication MOM (Message-Oriented Middleware) monitors TP monitors motivation 2nd 3rd motivation for creating SOAs 2nd 3rd 4th agility 2nd 3rd business infrastructure 2nd cost savings 2nd at business level 2nd IT 2nd 3rd efficient development processes evolutionary approach 2nd 3rd feedback from 2nd independence from technology 2nd 3rd mitigating risk 2nd 3rd 4th 5th reuse 2nd MQSeries 2nd MS Visual Basic VBX components multi-channel applications 2nd 3rd fundamental SOA 2nd process-enabled SOAs 2nd 3rd 4th 5th . service facades 2nd 3rd multichannel architecture multilevel transactions multimedia message service (MMS) Myers mySAP

### Team LiB



## **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

▲ PREVIOUS NEXT ►

## Index

 $[\underline{\mathsf{SYMBOL}}][\underline{\mathsf{A}}][\underline{\mathsf{B}}][\underline{\mathsf{C}}][\underline{\mathsf{D}}][\underline{\mathsf{E}}][\underline{\mathsf{F}}][\underline{\mathsf{G}}][\underline{\mathsf{H}}][\underline{\mathsf{I}}][\underline{\mathsf{I}}][\underline{\mathsf{K}}][\underline{\mathsf{L}}][\underline{\mathsf{M}}][\underline{\mathsf{M}}][\underline{\mathsf{O}}][\underline{\mathsf{P}}][\underline{\mathsf{Q}}][\underline{\mathsf{R}}][\underline{\mathsf{S}}][\underline{\mathsf{T}}][\underline{\mathsf{U}}][\underline{\mathsf{V}}][\underline{\mathsf{W}}][\underline{\mathsf{X}}]$ 

Naming Services ORB nested transactions NetWare Loadable Modules (NLM) network SOA networked SOA 2nd 3rd 4th 5th 6th 7th NLA (Non Life Applications) NLM (NetWare Loadable Modules) Non Life Applications (NLA) Norwegian Computing Center Novell NetWare Loadable Modules (NLM) Nygaard, Kristen

### Team LiB

#### ♦ PREVIOUS NEXT ▶



## **ABC Amber CHM Converter Trial vers**

Please register to remove this banner.

#### Team LiB

♦ PREVIOUS NEXT ►

## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

Object Management Group (OMG) object orientation Object Request Broker [See ORB] object-oriented programming 2nd obiects off-the-shelf load balancers interoperability offsetting Intelligent Finance OLE\_LINK1 OLTP (Online Transaction Processing) OMG CORBA OMG (Object Management Group) Online Transaction Processing [See OLTP] operating systems UNIX operations update operations 2nd 3rd error handling 2nd sequence numbers 2nd optimisitc concurrency control 2nd 3rd 4th 5th example 2nd 3rd 4th 5th Oracle ORB Distributed Objects Naming Services ORB (Object Request Broker) organizational challenges limitations of 2PC limitations of ACID transactions organizational roadmaps 2nd organizational SOA roadmaps 2nd 3rd 4th organizing IT 2nd orientation service orientation out of stock exceptions outside intervention

#### Team LiB

#### ◀ PREVIOUS NEXT ►



Team LiB

+ PREVIOUS NEXT +

## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

parallel iterations managing 2nd 3rd Pascal passenger check-in scenario 2nd 3rd payload semantics 2nd coupling versus interface semantics 2nd 3rd document-centric messages 2nd peer-programming PeopleSoft performance limitations of 2PC limitations of ACID transactions persistent queues 2nd pessimistic concurrency control example 2nd 3rd PIMs (Platform Independent Models) Platform Independent Models (PIMs) Platform Specific Models [See PSMs] presentation layer process and desktop intergration process control logic process inconsistencies process integrity 2PC 2nd ACID transactions 2nd 3rd BPM 2nd **BPMs** distributed 2PC logging and tracing 2nd 3rd multilevel transactions nested transactions persistent queues 2nd SAGAs 2nd SOA-driven project management 2nd 3rd 4th transaactional steps 2nd transaction chains 2nd transaction monitors 2nd 3rd versus data integrity Web service standards 2nd 3rd process laye process logic 2nd 3rd process management process orientation BPM process-centric service process-centric services 2nd 3rd 4th process-enabled SOAs 2nd 3rd 4th 5th 6th 7th 8th BPM 2nd core business logic versus process control logic 2nd 3rd 4th design implications 2nd multi-channel applications 2nd 3rd 4th 5th processes implementing SOA at Deutsche Post 2nd 3rd 4th 5th 6th implementing SOA at CSG 2nd 3rd 4th implementing SOA at Winterther 2nd 3rd in an ideal world 2nd 3rd 4th 5th 6th Product Lifecycle Management products programming paradigms component programming functional decomposition functional programming object-oriented programming 2nd service-orientation project control elements SOA artifacts 2nd 3rd 4th project definitions including service designs 2nd 3rd project management implementing at Intelligent Finance 2nd 3rd architecture boards DIA IT steering committee work streams XML tsars 2nd 3rd implementing SOA at Deutsche Post 2nd implementing SOA at CSG 2nd 3rd implementing SOA at Winterthur 2nd project management methodologies adding service orientation to configuration management challenges for 2nd 3rd recommendations for the SOA integration 2nd 3rd 4th 5th establishing 2nd 3rd 4th SOA-driven project management 2nd 3rd 4th including service designs in the 2nd 3rd leveraging SOA to decompose 2nd 3rd 4th 5th 6th leveraging SOA to drive 2nd 3rd 4th 5th 6th 7th

process integrity 2nd 3rd 4th

#### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

▲ PREVIOUS NEXT ►

## Index

 $[\underline{\mathsf{SYMBOL}}][\underline{\mathsf{A}}][\underline{\mathsf{B}}][\underline{\mathsf{C}}][\underline{\mathsf{D}}][\underline{\mathsf{E}}][\underline{\mathsf{F}}][\underline{\mathsf{G}}][\underline{\mathsf{H}}][\underline{\mathsf{I}}][\underline{\mathsf{I}}][\underline{\mathsf{K}}][\underline{\mathsf{L}}][\underline{\mathsf{M}}][\underline{\mathsf{M}}][\underline{\mathsf{O}}][\underline{\mathsf{P}}][\underline{\mathsf{Q}}][\underline{\mathsf{R}}][\underline{\mathsf{S}}][\underline{\mathsf{T}}][\underline{\mathsf{U}}][\underline{\mathsf{V}}][\underline{\mathsf{V}}][\underline{\mathsf{M}}][\underline{\mathsf{X}}]$ 

QoS (Quality of Service) Quality of Service [See QoS] <u>queues</u> persistent queues 2nd

Team LiB

♦ PREVIOUS NEXT ►



#### Team LiB

◀ PREVIOUS NEXT ►

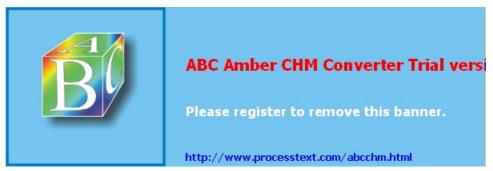
## Index

[<u>SYMBOL</u>] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [**R**] [S] [T] [U] [V] [W] [X]

r-tools suite R/2 RAD (Rapid Application Development) Rapid Application Development (RAD) RAS (Reliability/Availability/Serviceability) read stability real world SOAs example of failure 2nd 3rd 4th example of success 2nd 3rd recommendations for SOA integration team 2nd 3rd 4th 5th for SOA protagonists 2nd 3rd reducing risk 2nd Reengineering the Corporation (s/b ital) refactoring enterprise software architecture referential integrity regression test environments relational database Reliability/Availability/Serviceability [See RAS] remote procedure call system SUN-RPC standard Remote Procedure Calls [See RPCs] remoteness Rendezvous repeatable read repositories CSG Deutsche Post case study implementing at Intelligent Finance 2nd implementing SOA at CSG 2nd implementing SOA at Winterthur 2nd Intelligent Finance Winterthur 2nd 3rd requirements of enterprise software architecture 2nd 3rd 4th return on investment (ROI) reusability requirements of enterprise software architecture reuse 2nd 3rd risk reducing 2nd risk analysis risk-mitigating effect motivation for creating SOAs 2nd 3rd 4th 5th roadmaps enterprise IT renovation roadmap [See enterprise IT renovation roadmap] organizational roadmaps 2nd organizational SOA roadmaps 2nd 3rd 4th technical roadmaps ROI (return on investment) RosettaNet **RPC-style** interfaces RPCs 2nd 3rd fire-and-forget RPCs (Remote Procedure Calls) runtime binding 2nd runtime configuration runtime service discovery based on reflection runtime service lookup by name runtime service lookup by properties

#### Team LiB

♦ PREVIOUS NEXT ►



Team LiB

+ PREVIOUS NEXT +

# Index

service registry implementing SOA

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [**S**] [T] [U] [V] [W] [X] SAGAs 2nd SAML (Security Assertion Markup Language) SAP 2nd R/2 SBB Deutsche Post SBB (Service Backbone) scalability 2nd 3rd 4th 5th 6th CICS 2nd CORBA EJBs 2nd 3rd 4th in a heterogeneous SOA 2nd Web Services 2nd 3rd wrapped legacy applications 2nd SCM (Supply Chain Management) 2nd securing SOAs 2nd 3rd 4th 5th authentication 2nd 3rd 4th 5th 6th authorization 2nd 3rd 4th 5th 6th encryption 2nd 3rd transport security 2nd 3rd trust domains 2nd 3rd security and heterogeneity 2nd 3rd 4th 5th CSG 2nd Deutsche Post case study execution containers J2ME MIDP lightweight security Winterthur 2nd 3rd Security Assertion Markup Language (SAML) security infrastructures B2B security solution semi-transactional steps 2nd 3rd 4th separating SOA services server controlled transactions 2nd 3rd service 2nd 3rd 4th basic services data-centric services 2nd 3rd 4th logic-centric services 2nd 3rd business computing [See business computing] cost effectiveness distributed computing 2nd 3rd 4th 5th 6th SOAP distributed computting XML intermediary services adapters facades 2nd 3rd 4th functionality-adding services 2nd technology gateways 2nd Log Services Naming Services ORB process-centric services 2nd 3rd 4th SOA [See SOA] Web Services World Wide Web service access layer Service Backbone (SBB) service binding service bus SOA 2nd service clients exposing to transaction logic service contracts 2nd service contract iterations service designs including in project definitions 2nd 3rd service dispatchers service documentation service enablement EAI 2nd 3rd 4th 5th service facades multi-channel applications 2nd 3rd service interfaces CSG Deutsche Post case study Intelligent Finance 2nd versus services 2nd Winterthur 2nd 3rd service layers creating layers that replace direct interaction with distributed objects Intelligent Finance 2nd 3rd service orientation adding to project management methodologies Service Registry

#### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

Team LiB

+ PREVIOUS NEXT +

## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

```
teams
   success of SOAs
technical failures
technical integration
technical roadmaps
technical services
technology
   complexity
   CSG
   architecture 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th
Deutsche Post
      architecture 2nd 3rd 4th
      contracts
      management
     repositories
     security
      service interfaces
   Intelligent Finance
      architecture 2nd
      contracts
     repositories
      service interfaces 2nd
   Winterthur 2nd 3rd 4th
     contracts 2nd 3rd
      management 2nd 3rd
     repositories 2nd 3rd
      security 2nd 3rd
     service interfaces 2nd 3rd
technology gateways 2nd
technology impact
CSG 2nd 3rd 4th 5th
   Deutsche Post 2nd
   Intelligent Finance 2nd 3rd 4th 5th 6th
   Winterthur 2nd 3rd 4th 5th 6th
technology whitepapers
telnet
test drivers
test suites
   creating
testing 2nd 3rd 4th 5th 6th 7th 8th
   functional testing
  regression test environments
      creating
   systematic testing
  test suites
creating
thin thread model 2nd 3rd 4th
thin-thread approach
Tibco Software
   Rendezvous
tight coupling 2nd 3rd 4th 5th 6th
timestamps
   optimistic concurrency control
tokens
   session-tokens
   transaction-tokens
tools
   automated test tools
top-down code generation 2nd
Total Quality Management
TP monitors [See transaction monitors]
TPMs [See transaction monitors]
TPMs (Transaction Processing Monitors)
tracing
transaction boundaries
logging 2nd 3rd
transaction chains 2nd
   combining with compensating transactions
transaction coordinator
transaction logic
   exposing to service clients
transaction logs
transaction management
   execution containers
transaction monitors 2nd 3rd 4th 5th 6th
   logging
Transaction Processing Monitors [See TPMs]
transaction-tokens
transactional steps 2nd 3rd 4th 5th 6th 7th 8th 9th
   choosing granularity 2nd 3rd 4th
   semi-transactional steps 2nd 3rd 4th
transactions
   ACID transactions 2nd 3rd
   client controlled transactions 2nd 3rd
   compensating transactions 2nd
   multilevel transactions
   nested transactions
   server controlled transactions 2nd 3rd
   Web services-based transaction protocols
transport security 2nd 3rd
tresource managers
trust domains 2nd 3rd 4th
```

#### Team LiB



### **ABC Amber CHM Converter Trial versi**

Please register to remove this banner.

#### Team LiB

▲ PREVIOUS NEXT ►

## Index

 $[\underline{\mathsf{SYMBOL}}][\underline{\mathsf{A}}][\underline{\mathsf{B}}][\underline{\mathsf{C}}][\underline{\mathsf{D}}][\underline{\mathsf{E}}][\underline{\mathsf{F}}][\underline{\mathsf{G}}][\underline{\mathsf{H}}][\underline{\mathsf{I}}][\underline{\mathsf{J}}][\underline{\mathsf{K}}][\underline{\mathsf{L}}][\underline{\mathsf{M}}][\underline{\mathsf{M}}][\underline{\mathsf{O}}][\underline{\mathsf{P}}][\underline{\mathsf{Q}}][\underline{\mathsf{R}}][\underline{\mathsf{S}}][\underline{\mathsf{T}}][\underline{\mathsf{U}}][\underline{\mathsf{V}}][\underline{\mathsf{W}}][\underline{\mathsf{X}}]$ 

Uber bus UDDI UDDI (Universal Description, Discovery and Integration UN/CEFACT (ebXML) uncommited read Universal Description, Discovery and Integration [See UDDI] UNIX workstations update operations 2nd 3rd error handling 2nd sequence numbers 2nd upgrade ability EAI 2nd 3rd 4th user-defined data integrity users proxy users

Team LiB

♦ PREVIOUS NEXT ▶



### **ABC Amber CHM Converter Trial vers**

Please register to remove this banner.

#### Team LiB

▲ PREVIOUS NEXT ►

## Index

 $[\underline{\mathsf{SYMBOL}}][\underline{\mathsf{A}}][\underline{\mathsf{B}}][\underline{\mathsf{C}}][\underline{\mathsf{D}}][\underline{\mathsf{E}}][\underline{\mathsf{F}}][\underline{\mathsf{G}}][\underline{\mathsf{H}}][\underline{\mathsf{I}}][\underline{\mathsf{J}}][\underline{\mathsf{K}}][\underline{\mathsf{L}}][\underline{\mathsf{M}}][\underline{\mathsf{M}}][\underline{\mathsf{O}}][\underline{\mathsf{P}}][\underline{\mathsf{Q}}][\underline{\mathsf{R}}][\underline{\mathsf{S}}][\underline{\mathsf{T}}][\underline{\mathsf{U}}][\underline{\mathsf{V}}][\underline{\mathsf{W}}][\underline{\mathsf{X}}]$ 

VBX components vendors of standard software perspective of SOAs version counts optimistic concurrency control vertical slicing versus horizontal slicing 2nd

vision of BPM 2nd 3rd 4th 5th VT100 systems

#### Team LiB

#### ♦ PREVIOUS NEXT ►



### **ABC Amber CHM Converter Trial versi**

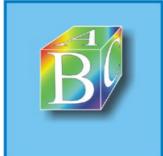
Please register to remove this banner.

#### Team LiB

◀ PREVIOUS NEXT ►

#### Index [SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [**W**] [X] Wal-Mart business computing Web applications building 2nd 3rd 4th 5th 6th 7th 8th Web Service Definition Language [See WSDL] Web service standards process integrity 2nd 3rd Web Services availability 2nd 3rd scalability 2nd 3rd whitepapers 2nd business whitepapers technology whitepapers wincoLink Winterthur 2nd 3rd 4th implementing SOA 2nd processes and structures 2nd 3rd project management 2nd repositories 2nd lessons learned from implementing SOA 2nd 3rd 4th project scope business impact 2nd technology impact 2nd 3rd 4th 5th 6th technology 2nd 3rd 4th contracts 2nd 3rd management 2nd 3rd repositories 2nd 3rd security 2nd 3rd service interfaces 2nd 3rd Wintherthur Wirth, Niklaus Pascal WMS (Workflow Management System) work streams workflow components Workflow Management Systems [See WMS] workstations World Wide Web service wrapped legacy aplications availability 2nd wrapped legacy applications scalability 2nd write-test-cases-before-writing-the-actual-code WSDL (Web Service Definition Language) Team LiB

#### ♦ PREVIOUS NEXT ▶



## ABC Amber CHM Converter Trial versi

Please register to remove this banner.

#### Team LiB

4 PREVIOUS

## Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

X/Open DTP (X/Open standard for Distributed Transaction Processing) X/Open standard for Distributed Transaction Processing [See X/Open DTP] XA interface XML implementing at Intelligent Finance 2nd XML tsars 2nd 3rd

#### Team LiB

4 PREVIOUS

